MOSAIC Calculus

Daniel Kaplan

4/8/2022

Table of contents

Preface

Tips in draft

```
GIT_TRACE=1 git push
```

or

```
git push --force origin main
```

Welcome to calculus

Calculus is the set of concepts and techniques that form the mathematical basis for dealing with motion, growth, decay, and oscillation. The phenomena can be as simple as a ball arcing ballistically through the air or as complex as the airflow over a wing that generates lift. Calculus is used in biology and business, chemistry, physics and engineering. It is the foundation for weather prediction and understanding climate change. It is the basis for the algorithms for heart rate and blood oxygen measurement by wristwatches. It is a key part of the language of science. The electron orbitals of chemistry, the stresses of bones and beams, and the business cycle of recession and rebound are all understood primarily through calculus.

Calculus has been central to science from the very beginnings. It is no coincidence that the scientific method was introduced and the language of calculus was invented by the same small group of people during the historical period known as the ***Enlightenment***. Learning calculus has always been a badge of honor and an entry ticket to professions. Millions of students' career ambitions have been enhanced by passing a calculus course or thwarted by lack of access to one.

In the 1880s, a hit musical featured "the very model of a modern major general." One of his claims for modernity: "I'm very good at integral and differential calculus." (Watch here.)

What was modern in 1880 is not modern anymore. Yet, amazingly, calculus today is every bit as central to science and technology as it ever was and is much more important to logistics, economics and myriad other fields than ever before. The reason is that science, engineering, and society have now fully adopted the computer for almost all aspects of work, study, and life. The collection and use of data is growing dramatically. Machine learning has become the way human decision makers interact with such data.

Think about what it means to become "computerized." To take an everyday example, consider video. Over the span of a human life, we moved from a system which involved people going to theaters to watch the shadows recorded on cellulose film to the distribution over the airwaves by low-resolution television, to the introduction of high-def broadcast video, to on demand streaming from huge libraries of movies. Just about anyone can record, edit, and distribute their own video. The range of topics (including calculus) on which you can access a video tutorial or demonstration is incredibly vast. All of this recent progress is owed to computers.

The "stuff" on which computers operate, transform, and transmit is always mathematical representations stored as bits. The creation of mathematical representations of objects and events in the real world is essential to every task of any sort that any computer performs. Calculus is a key component of inventing and using such representations.

You may be scratching your head. If calculus is so important, why is it that many of your friends who took calculus came away wondering what it is for? What's so important about "slopes" and "areas" and how come your high-school teacher might have had trouble telling you what calculus is for?

The disconnect between the enthusiasm expressed in the preceding paragraphs and the lived experience of students is very real. There are two major reasons for that disconnect, both of which we tackle head-on in this book.

First, teachers of mathematics have a deep respect for tradition. Such respect has its merits, but the result is that almost all calculus is taught using methods that were appropriate for the era of paper and pencil–not for the computer era. As you will see, in this book we express the concepts of calculus in a way that carries directly over to the uses of calculus on computers and in genuine work.

Second, the uses of calculus are enabled not by the topics of Calc I and Calc II alone, but the courses for which Calc I/II are a preliminary: linear algebra and dynamics. Only a small fraction of students who start in Calc I ever reach the parts of calculus that are the most useful. Fortunately, there is a large amount of bloat in the standard textbook topics of Calc I/II which can be removed to make room for the more important topics. We try to do that in this book.

Computing and apps

The text provides two complementary ways to access computing. The most intuitive is designed purely to exercise and visualize mathematical concepts through mouse-driven, graphical ***apps***. To illustrate, here is an app that we'll use in Block 6. You can click on the snapshot to open the app in your browser.

More fundamentally, you will be carrying out computing by composing computer commands and text and having a computer carry out the commands. One good way to do this is in a ***sandbox***–a kind of app which provides a safe place to enter the commands. You'll access the sandbox in your browser (click on the image below to try it now).

Once you've entered the computer commands, you press the "Run" button to have the commands carried out. (You can also press CTRL+Enter on your keyboard.)

An important technique for teaching and learning computing is to present ***scaffolding*** for computer commands. At first, the scaffolding may be complete, correct commands that can be cut-and-pasted into a ***sandbox*** where the calculation will be carried out. Other times it will be left to the student to modify

or fill in missing parts of the scaffolding. For example, when we introduce drawing graphs of functions and the choice of a domain, you might see a scaffold that has blanks to be filled in:

```
slice_plot( exp(-3*t) ~ t, domain( --fill in domain-- ))
```

You can hardly be expected at this point to make sense of any part of the above command, but soon you will.

After you get used to computing in a sandbox, you may prefer to install the R and RStudio software on your own laptop. This usually provides a faster response to you and lowers the load on the sandbox cloud servers being used by other students.

Experienced R users may even prefer to skip the sandbox entirely and use the standard resources of RStudio to edit and evaluate their computer commands. You'd use exactly the same R commands regardless of whether you use a cloud server or your own laptop.

Exercises and feedback

Learning is facilitated by rapid, formative feedback. Many of the exercises in this book are arranged to give this.

LINK TO AN EXERCISE HERE

Practice, practice, practice

It's a good practice to practice! The Drill app provides multiple-choice questions designed to be answered at a glance or a very small amount of work on scratch paper. Once you choose a topic, the questions are presented in random order. You get immediate feedback on your answer. If your answer was wrong, the question is queued up again so that you'll have another chance. At the point where you are answering almost all questions correctly, you're ready to move on.

Software for the course

You can get started with the course using just a web browser. In addition to this textbook, bookmark the SANDBOX and DRILL QUESTIONS so you can get to them easily.

If you find that the web sites are too slow, you can install both the sandbox and drill apps on your own computer. (You'll need a computer running Windows or OS-X or Linux. Smartphones or tablets won't let you do this.)

If you already use RStudio, you can skip to step (3). You can use the *MOSAIC Calculus* software directly from RStudio as an alternative to the sandbox. See step (5).

Here are the steps. Steps (1), (2), and (3) together will take almost half an hour. Once they are completed, you will not need to do them again on that computer.

If you already have R and RStudio installed, skip to Step (3).

1. Install the R software. You can find reasonable video instructions on the Internet, for instance at YouTube
   - R installer for Windows.
   - R installer for OS-X
2. Install RStudio RStudio installer

Not everyone has full permission to install external apps on their laptop. This is particularly true when the computer has been issued by your educational institution. If you are in this situation or, for other reasons, can't complete steps (1) and (2) completely, seek help from a local expert. Both (1) and (2) have been installed by students on tens of millions of computers.

3. Install the MOSAIC Calculus packages within R. Launch the RStudio app, just as you would launch any other app.
   - When the RStudio app starts, the upper left pane will be labeled "Console" and there will be a prompt: >
   - Copy and paste these commands, one at a time, after the console prompt, pressing return after each command:

```
install.apps(c("remotes", "distillr"))
remotes::install_github("dtkaplan/Zcalc")
```

4. On a daily basis, whenever you need to use the *MOSAIC Calculus* software.

   a. Open the RStudio App in the usual way for your operating system.
   b. In the RStudio console, give these two commands after the console prompt:

```
library(Zcalc)
Sandbox()
```

The first command will take about 15 seconds to complete. It will display some messages in the Console, which you can ignore. **NOTE**: If you are given a message asking if you want to install a "personal" or "private" library, say yes. But if you are a system used by multiple people who need the software, select the option to install for all users.

The second command will take about 5 minutes. Hundreds of [illegible] messages will appear in the Console, which you can ignore. (If you are working from a previous installation of R, you may be asked to update existing packages. Since you can do this at any time in the future, decline the update now.)

The computing sandbox will open in a browser tab. Closing that tab will return control to the console.

When you want to practice with the drill questions for this book, give the command `Drill()` instead of `Sandbox()`.

Most people find it convenient, when using the software several times a week, simply to keep the RStudio session open and similarly with the browser tab with the Sandbox.

5. **Not required**. Many students are taught how to use RStudio directly. If you are in this situation, you will be able to take advantage of the many features provided by this sophisticated software. There are two things to keep in mind:

   i. At the start of an RStudio session, give the command `library(Zcalc)` in the console.
   ii. If you are writing RMarkdown documents, then the following should make sense to you: Include `library(Zcalc)` in the start-up chunk so that it will run whenever you compile your document.

Video resources

We're constructing a list to some of the videos we have found that can be useful in solidifying your understanding of calculus concepts.

Suggestion are most welcome. Email a link to `dtkaplan@gmail.com`

Block 2

- Derivatives as measuring stretching and shrinking: 3Blue1Brown
- Derivatives of power-law functions: 3Blue1Brown
- Derivative of sinusoids: 3Blue1Brown
- Derivatives of sums, products and compositions 3Blue1Brown

Block 5

- Flatland from Karl Sagan
- Fourier and Laplace transforms intuition

Block 6

- Dynamics of exponential and limited growth. [3Blue1Brown]
- Modeling epidemics with differential equations 3Blue1Brown
- Forcing an oscillator Mathematics of vibration

Acknowledements

This project was initiated by the Mathematical Sciences department at the US Air Force Academy. They recognized that a traditional calculus introduction is ill-suited to the needs of STEM in the 21st century.

Critical support was given by the ARDI Foundation which awarded the Holland H. Coors Chair in Education Technology to one of the project members, Daniel Kaplan. This made possible a year-long residency at USAFA during which time he was able to work unhindered on this project.

Macalester College, where Kaplan is DeWitt Wallace Professor of Mathematics, Statistics, and Computer science, was the site where the overall framework and many of the materials for a STEM-oriented calculus were developed. Particularly important in the germination were David Bressoud and Jan Serie, respectively chairs of the Macalester math and biology departments, as well as Prof. Thomas Halverson and Prof. Karen Saxe, who volunteered to team teach with Kaplan the first prototype course. Early grant support from the Howard Hughes Medical Foundation and the Keck Foundation provided the resources to carry the prototype course to a point of development where it became the entryway to calculus for Macalester students.

Profs. Randall Pruim (Calvin University) and Nicholas Horton (Amherst College) were essential collaborators in developing software to support calculus in R. They and Kaplan formed the core team of Project MOSAIC, which was supported by the US National Science Foundation (NSF DUE-0920350).

Joel Kilty and Alex McAllister at Centre College admired the Macalester course and devoted much work and ingenuity to write a textbook, *Mathematical Modeling and Applied Calculus* (Oxford Univ. Press), implementing their own version. Their textbook enabled us to reduce the use of sketchy notes in the first offering of this course at USAFA.

To learn more about Quarto books visit https://quarto.org/docs/books.

Put PDF into Tufte mode.

```
library(ggplot2)
mtcars2 <- mtcars
mtcars2$am <- factor(
  mtcars$am, labels = c('automatic', 'manual')
)
ggplot(mtcars2, aes(hp, mpg, color = am)) +
  geom_point() + geom_smooth() +
  theme(legend.position = 'bottom')
```

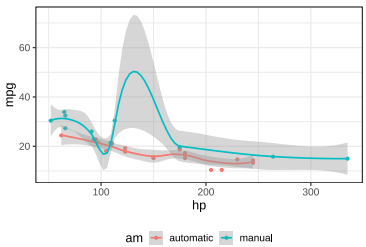


Figure 0.1: MPG vs horsepower, colored by transmission.

Theorem 0.1. *The first theorem is this.*

You've got to have a `data-latex=""` for the latex environment to be invoked.

Why did you?

Calling the booboo environment from a div

Part I

Preliminaries

This is where I'll explain what the block is about and the overall goes.

1 Modeling change

Calculus is about change, and change is about relationships. Consider the complex and intricate network of relationships that determine climate: a changing climate implies that there is a relationship between, say, global average temperature and time. Scientists know temperature changes with levels of CO_2 and methane which themselves change due to their production or elimination by atmospheric and geological processes. A change in one component of climate (e.g., ocean acidification or pH level) provokes change in others.

In order to describe and use the relationships we find in the natural or designed world, we build mathematical representations of them. We call these ***mathematical models***. On its own, the word "model" signifies a representation of something in a format serves a specific ***purpose***. A blueprint describing the design of a building is an everyday example of a model. The blueprint *represents* the building but in a way that is utterly different from the building itself. Blueprints are much easier to construct or modify than buildings, they can be carried and shared easily. Two of the purposes of a blueprint is to aid in the design of buildings and to communicate that design to the people securing the necessary materials and putting them together into the building itself.

Atmospheric scientists build models of climate whose purpose is to explore scenarios for the future emission of greenhouse gasses. The model serves as a stand-in for the Earth, enabling predictions in a few hours of decades of future change in the climate. This is essential for the development of policies to stabilize the climate.

Designing a building or modeling the climate requires expertise and skill in a number of areas. Nonetheless, constructing a model is *relatively easy* compared to the alternative. Models

also make it relatively easy to extract the information that's needed for the purpose at hand. For instance, a blueprint gives a comprehensive overview of a building in a way that's hard to duplicate just by walking around an actual building.

A ***mathematical model*** is a model made out of mathematical and computational stuff. Example: a bank's account books are a model made mostly out of numbers. But in technical areas—science and engineering are obvious examples, but there are many other fields, too—numbers don't get you very far. By learning calculus, you gain access to important mathematical and computational concepts and tools for building models and extracting information from them.

A major use for mathematics is building models, representation for a purpose, constructed out of mathematical things some of which we describe in the next section.

Models are easy to manipulate compared to reality, easy to implement (think "draw a blueprint" versus "construct a building"), and easy to extract information from. We can build multiple models and compare and contrast them to gain insight to the real-world situation behind the models.

This book presents calculus in terms of two simple concepts central to the study of change: ***quantities*** and ***functions***. Those words have everyday meanings which are, happily, close to the specific mathematical concepts that we will be using over and over again. Close … but not identical. So, pay careful attention to the brief descriptions that follow.

1.1 Quantity vs number

A mathematical ***quantity*** is an amount. How we measure amounts depends on the kind of stuff we are measuring. The real-world stuff might be mass or time or length. It equally well can be velocity or volume or momentum or corn yield per acre. We live in a world of such stuff, some of which is tangible (e.g., corn, mass, force) and some of which is harder to get your hands on and your minds around (acceleration, crop yield, fuel economy). An important use of calculus is helping us

conceptualize the abstract kinds of stuff as mathematical compositions of simpler stuff. For example, crop yield incorporates mass with length and time. Later, you'll see us using the more scientific-sounding term ***dimension*** instead of "stuff." In fact, Chapter @ref(dimensions) is entirely dedicated to the topic of dimensions, but for now it's sufficient for you to understand that numbers alone are not quantities.

Most people are inclined to think "quantity" is the same as "number"; they conflate the two. This is understandable but misguided. By itself a number is meaningless. What meaning does the number 5 have without more context? Quantity, on the other hand, combines a number with the appropriate context to describe some amount of stuff.

The first thing you need to know about any quantity is the kind of stuff it describes. A "mile" is a kind of stuff: length. A meter is the same kind of stuff: length. A liter is a different kind of stuff: volume. A gallon and an acre-foot are the same kind of stuff: volume. But an inch (length) is not the same kind of stuff as an hour (time).

"Stuff," as we mean it here, is what we measure. As you know, we measure with ***units***. Which units are appropriate depends on the kind of stuff. Meters, miles, microns are all appropriate units of length, even though the actual lengths of these units differ markedly. (A mile is roughly 1.6 million millimeters.)

Only after you know the dimension and units does the number have meaning. Thus, a *number* is only part of specifying a *quantity*.

Here's the salient difference between number and quantity when it comes to calculus: All sorts of arithmetic and other mathematical operations can be performed to combine numbers: addition, multiplication, square roots, etc. When performing mathematics on quantities, only multiplication and division are universally allowed. For addition and subtraction, square roots, and such, the operation makes sense only if the dimensions are suitable.

The mathematics of units and dimension are to the technical world what common sense is in our everyday world. For instance (and this may not make sense at this point), if people

tell me they are taking the square root of 10 liters, I know immediately that either they are just mistaken or that they haven't told me essential elements of the situation. It's just as if someone said, "I swam across the tennis court." You know that person either used the wrong verb—walk or run would work—or that it wasn't a tennis court, or that something important was unstated, perhaps, "During the flood, I swam across the tennis court."

1.2 Functions

Functions, in their mathematical and computing sense, are central to calculus. At the start of the chapter, it says, "Calculus is about change, and change is about relationships." The idea of a mathematical function gives a definite perspective on this. The relationship represented by a function is between the function's input and the function's output. The input might be day of the year (1-365, often called the "Julian Date"), and the output cumulative rainfall up to that day. Every day it rains, the cumulative rainfall increases. Another relationship another function: the input might be the altitude on your hike up Pikes Peak and the output the air temperature. Typically, as you gain altitude the temperature goes down. With still another function, the input might be the number of hours after noon, the output the brightness of sunlight. As the sun goes down, the light grows dimmer, but only to a point.

A function is a mathematical concept for taking one or more ***inputs*** and returning an ***output***. In calculus, we'll deal mainly with functions that take one or more quantities as inputs and return another quantity as output. But sometimes we'll work with functions that take functions as input and return a quantity as output. And there will even be functions that take a function as an input and return a function as output.

You've almost certainly seen functions expressed in the mathematical form $f(x)$. The function is $f()$, but what is x? In high-school you likely learned to call x a "variable." This is standard in mathematics education but it is also the source of considerable confusion. To avoid that confusion, we are going

to be more precise about what x means. Try to put the word "variable" out of mind for the present, until we get to discussing the nature of data.

We will make extensive use of the term ***input***. So far as $f(x)$ is concerned, we will say that x is the ***name of an input***. $f(x)$ has only one input but soon we will work with functions that have multiple inputs.

The name x refers to something: the set of legitimate inputs to $f()$. For example, a function like sin() only accepts pure numbers as its input; quantities such as "3 meters" are not legitimate inputs because they have units and dimension. The word ***domain*** is a more concise way of saying "the set of legitimate inputs," as in "the domain of sin() is the set of real numbers." (***Real numbers*** is just a mathematical way of saying the values represented by the number line.)

Another important concept is ***applying a function*** to an input to produce an ***output***. For example, when we write sin(7.3) we give the numerical value 7.3 to the sine function. The sine function then does its calculation and returns the value 0.8504366. We generally prefer to write sin(7.3) rather than 0.8504366 for reasons of communication. When a person sees sin(7.3) he or she is reminded of the motivation the modeler had in mind for specifying that particular value.

Other ways that we'll signal that we are applying a function to an input is by writing something like $\sin(x = 7.3)$ or, later in the book, $\sin(x)\Big|_{x=7.3}$.

Just as a "domain" is the set of legitimate inputs to a function, the function's ***range*** is the set of values that the function can produce as output. For instance, the range of sin() is the numbers between -1 and 1 which we'll usually write in this format: $-1 \leq x \leq 1$.

TAKE NOTE!

In high-school, you may have seen an expression like $mx + b$. If so, you learned to call it "a line" or perhaps even "a function."

The proper term for it is a ***formula***. Formulas are one way of describing how to do a calculation.

You may also have seen an expression like $y = mx + b$. This is, of course, an ***equation***, but equations are massively overused in mathematics education. An equation like this is typically used to signify that "y is a function of x," but we are going to be diligent in making explicit when we are defining a function. We will write

$$f(x) \equiv mx + b$$

to mean "we define a function named $f()$ that takes an input named x." The formula on the right side of $\equiv$ tells how $f()$ calculates the value of the output for any given input.

What's wrong with writing an equation like $y = mx + b$ to define a function? The nature of equations is that they can be re-arranged. For example

$$y = mx + b \text{ might be re-arranged as } \quad m = \frac{y - b}{x} \ .$$

Two different equations expressing the same relationship. But as definitions of functions the two equivalent equations mean might mean two different things: $y()$ takes x as an input or, quite differently, $m()$ takes both x and y as inputs. Much better, for all concerned, to define a function

$$\text{line}(x) = mx + b$$

which has a name (line()) for the function and a name x for the input. It also suggests that m and b are not inputs. You may already know that quantities like m and b, if not explicitly given as inputs, are called ***parameters***.

Another problem with $y = mx + b$ is that in almost all computer languages it means something completely different than than the definition of a function. Since you will be working with computers extensively in your career, we want to have a mathematical notation that is compatible with computer notation.

You will need to get used to this idea of defining a function and naming the inputs explicitly, but it will make your study of calculus much more useful.

The engineers and mathematicians who invented computer languages realized that they had to be explicit in identifying the input, the output, and the function itself; computers demand unambiguous instructions.[1] Sorting this out was a difficult process even for those mathematically talented and skilled pioneers of notation. So, you can be forgiven for the occasional confusion you have when dealing with notation that pre-dates computing.

In this book we'll be explicit and consistent in the ways we denote functions so that you can always figure out what are the inputs and how they are being translated into the output. A good start in learning to read the function notation is to see the equivalent of $y = mx + b$ in that notation:

$$g(x) \equiv mx + b$$

Math in the world ...

The various mathematical functions that we will be studying in this book are in the service of practical problems. But there are so many such problems, often involving specialized knowledge of a domain or science, engineering, economics, and so on, that a abstract mathematical presentation can seem detached from reality.

The video linked here, *How to shoot*, breaks down a simple-sounding situation into its component parts. The function itself is literally a black box. The inputs are provided by a human gunner training a telescope on a target and setting control dials. The ultimate output is the deflection of the guns in a remote turret. The main function is composed of several others, such as a function that outputs target range given the target size based on knowledge of the size of the target and how large it appears in the telescopic sight.

[1] Actually, it's common to give computers ambiguous instructions. The computer will carry out the instruction in the way it does, which may not be anything like what the programmer expected or intended.

Dividing the gunnery task into a set of inputs and a computed output allows for a division of labor. The gunner can provide the skills properly trained humans are good at, such as tracking a target visually. The computer provides the capabilities—mathematical calculation—to which electronics are well suited. Combining the inputs with the calculation provides an effective solution to a practical problem.

1.3 Spaces and change

The specialty of calculus is describing relationships between ***continuous sets***. Functions such as sin() or line(), which are typical of the functions we study in calculus, take numbers as input. A child learning about numbers starts with the "counting numbers": $1, 2, 3, ...$ In primary school, the set of numbers is extended to include zero and the negative numbers: $-1, -2, -3, ...$, giving a set called the ***integers***.

To almost everyone, its self-evident that the integers have an order and that the difference between successive integers is 1.

After this kids learn about "rational numbers," that is, numbers that are written as a ratio: $\frac{1}{2}, \frac{1}{3}, \frac{2}{3}, \ldots, \frac{22}{7}$, and so on. Rational numbers fit in the spaces between the integers.

If you didn't stumble on the word "spaces" in the previous sentence, you are well on your way to understanding what is meant by "continuous." For instance, between any two rational numbers there is another rational number. Think of the rational numbers as stepping stones that provide a path from any number to any other number.

Might a better analogy be a walkway instead of isolated stepping stones? A walkway is a structure on which you can move any amount, no matter how small, without risk of going off the structure. In contrast, a too-small move along a path of stepping stones will put you in the water.

A continuous set is like a walkway; however little you move from an element of the set you will still be on the set. The

(a) Discrete (b) Continuous

Figure 1.1: Discrete and continuous paths

continuous set of numbers is often called the ***number line***, although a more formal name is the ***real numbers***. ("Real" is a somewhat unfortunate choice of words, but we're stuck with it.)

The underlying metaphor here is space. Between any two points in space there is another point in space. We will have occasion to work with several different spaces, for instance:

- the number line: all the real numbers
- the positive numbers: the real numbers greater than zero
- the non-negative numbers: this is the tiniest little extension of the positive numbers adding zero to the set.
- a closed interval, such as the numbers between 5 and 10, which we will write like this: $5 \leq x \leq 10$, where x is a name we're giving to the set.
- the Cartesian plane: all pairs of real numbers such as $(5.62, -0.13)$. Other metaphors for this: the points on a piece of paper or a computer screen.
- three-dimensional coordinate spaces, generally written as a set of three real numbers such as $(-2.14, 6.87, 4.03)$ but really just the everyday three-dimension world that we live in.
- higher-dimensional spaces, but we won't go there until the last parts of the book.

Your spatial intuition of lines, planes, etc. will suffice for our needs. Mathematicians as a class value precise definitions; we won't need those. Widely accepted mathematical definitions of continuous sets date from the 1800s, 150 years *after* calculus was introduced. For instance, it's been known for more than

2000 years that there are numbers—the irrational numbers—that cannot be exactly expressed as a ratio of integers. We know now that there is an irrational number between any two rational numbers; the rational numbers are indeed analogous to stepping stones. But the distinction between rational and irrational numbers will not be one we need in this book. Instead, we need merely the notation of continuous space. And, by the way, the numbers stored and used in the normal way on computers are rational.

1.4 Exercises

Exercise XX.XX: YU5NCD unassigned

Exercise XX.XX: zbkNpV unassigned

Exercise 1.1: VDKUI unassigned

Exercise XX.XX: 37DC14 unassigned

2 Notation & computing

> *The ideas which are here expressed so laboriously are extremely simple …. The difficulty lies, not in the new ideas, but in escaping from the old ones, which [branch][1], for those brought up as most of us have been, into every corner of our minds.* — J. M Keynes, 1936, *The General Theory of Employment, Interest, and Money*, 1936

In addition to the specialized words we will use to express concepts and uses of calculus, we will also make extensive use of mathematical and computer-language notation. This chapter introduces you to the notation we'll be using.

One goal of good notation is to make clear which of these object types it is referring to. Another goal is to build on what you already know about how mathematics is written. For historical reasons these two goals are sometimes in conflict.

Yet another goal for notation has to with the central role of computing in the contemporary technical environment. Ideally, the mathematical notation we use should extend directly to computer-language notation. But in practice there is an incompatibility stemming from two sources:

1. Traditional mathematical notation makes extensive use of spatial arrangement, as for instance in $\frac{3}{4}$ or x^{-3} or $\sqrt[4]{y^2-6}$. For those familiar with it, this notation can be both concise and beautiful. But it was developed in an era of parchment and pen, without any inkling of keyboards and the strictly linear sequence of characters so widely used in written communication. Most mainstream computer languages are based on keyboard input.

[1] Original word: "ramify"

2. Traditional mathematical notation was developed for communicating between *people* and, like everyday language, has gaps and ambiguities that get sorted out (not always correctly) by human common sense. Computer languages, on the other hand, need to be precise, unambiguous, and interpreted by machines.

We'll attempt to use mathematical notation in a way that limits the conflict between tradition and computer notation. This conflict is particularly acute when it comes to the idea of an "equation," so widely used in high-school mathematics but not a component of mainstream computer languages.

2.1 Functions, inputs, and quantities

Our style of notation will be to give functions and their inputs *explicit names*. The basic principle is that a function name is a sequence of letters followed by an empty pair of parentheses, for instance sin() an ln().

Traditional mathematical notation writes many functions both without a name and without the parentheses. Examples that you have likely seen are x^2, $\sqrt{x}$, and e^x. If we were to absolutely impose the name/parentheses principle we would refer to these functions as, say, square() and sqrt() and exp(). Notice that the x is not part of the name.

Sometimes will will use names like square() just to emphasize the point that we are talking about a function. But for the most part we will stick to the traditional form because it is ubiquitous and recognizable by most readers.

The name/parentheses notation, like exp() or sin() allows us to avoid having to write x as the indicator of where the input to the function goes. That's helpful because, after all, the actual input might be something completely different from x.

Still, there are times in which we do need to state the name of the input to functions. One of these is when ***defining a function***. To define a function, we will use an expression like

$$g(y) \equiv y \cos(y) \ .$$

On the left of the $\equiv$ goes the name of the function, with the name of the input(s) in parentheses. On the right of $\equiv$ goes a formula for computing the output from the input. This formula is written in terms of the input name given on the left side of the definition.

In situations where there is just one input to a function, as in $g()$ above, we could use any name for the input. For instance, all of these are exactly equivalent to the definition for $g()$ given above:

$$g(x) \equiv x\cos(x) g(z) \equiv z\cos(z)) g(zebra) \equiv zebra\ \cos(zebra)$$

We'll tend to avoid hard-to-read input names like *zebra*. Instead, we'll mostly use :

- x or y or z.
- t. This name is typically used when the input is meant to be ***time***. So if we were creating a function to represent the relationship between time (of day) and outdoor brightness, we might use this notation: brightness(t)

Other input names we will use often in this book include u, v, w, following the 17th-century convention introduced by Newton that input names come from the end of the alphabet. But we won't shy away from more descriptive names, like T for "temperature" or V for volume, or even altitude (which describes itself).

When a function has more than one input, the input names serve to indicate where each input goes in the formula defining the calculation. For instance:

$$h(x, y) \equiv x^2 e^y \ .$$

$h()$ is a completely different function than, say, $f(x, y) \equiv y^2 e^x$.

You may have noticed that we've used the names $f()$, $g()$, and $h()$ a lot. Consider these names to be the equivalent of pronouns in English like "this", "that", "it", and so on. Function names

like $f()$ or $F()$ will be used when we need to refer to a function for a moment: a sentence, a paragraph, a section.

We will also have many occasions where we need to give a name to a quantity. Of course, a quantity is different from a function; functions are ***relationships*** between quantities.

For example, we will use names for quantities that are ***parameters*** in a function, like:

$$g(x) \equiv ax^2 + bx + c \ .$$

Here, x is the name given to the input to $g()$, while a, b, and c are names for other quantities involved in the formula.

Again following Newton's convention, names for quantities will come from the beginning of the alphabet. For instance, here is a definition of a function called a "cubic polynomial":

$$h(x) \equiv a + bx + cx^2 + dx^3 \ .$$

But there will be occasions where we need to compare two or more functions and run out of appropriate names from the start of the alphabet. A way to keep things organized is to use subscripts on the letters, for instancecomparing

$$g(x) \equiv a_0 + a_1x^2 + a_2x^2 + a_3x^3 + a_4x^4$$

to

$$f(x) \equiv b_0 + b_1x^2 + b_2x^2 \ .$$

Other ways professionals expand the set of letters from the start of the alphabet:

- Use capital letters: A, B, C, and so on
- Use Greek letters: α, β, γ, δ, …

2.2 Function output

We will often ***apply a function to*** specific input quantities in order to produce an output from the function. An equivalent phrase is ***evaluate a function on*** an input. For instance,

to apply the function $g()$ to the input quantity 3, any of the following mathematical expressions might be used:

$$g(3) \quad \text{or} \quad g(x = 3) \quad \text{or} \quad g(x)\Big|_{x=3} .$$

Remember that $g(3)$ or its equivalents are not themselves functions. They are the quantity that results from applying a the function to an input quantity.

2.3 Inputs, arguments, and variables

In everyday speech, an "argument" is a discussion between people with differing views. But in mathematics and computing, ***argument*** means something else entirely: it is a synonym for "input to a function."

In this text, we'll mostly use "input" to refer to what goes into a mathematical function, although using "argument" would be fine. As regards computer functions ... In Section @ref(makefun) you'll see how to instruct the computer to create a mathematical function like $g()$ or $f()$ from the previous section. The names and format of such instructions—e.g. make a mathematical function from a formula, draw a graph of a function, plot data—are given in the same function notation we use in math. For example, `makeFun()` constructs a function from a formula, `slice_plot()` graphs a function, `gf_point()` makes one style of data graphic. These R entities saying "do this" are also called "functions."

When referring to such R "do this" functions, we'll refer to the stuff that goes in between the parentheses as "arguments." The word "input" would also be fine. The point of using "input" for math functions and "argument" for R "do this" functions is merely to help you identify when we are talking about mathematics and when we are talking about computing.

A word we will **not** make much use of is "variable." You are probably used to statements like, "x and y are the variables," and it will take you a while to stop using them reflexively. The reason we will use "input" or "argument" instead of "variable" is that variable means too many different things in different

contexts. For instance, in the algebra-course instruction, "Solve $3x - 2 = x^2$," the x is really a quantity, unknown at first but soon to be resolved by your algebraic skills. The x in the solving problem would often be called a "variable," but it's not at all an "input" or an "argument."

There are two contexts in which we will use "variable," neither of which has to do with inputs to functions. In talking about data, we will use "variable" in the statistical sense, meaning "a type of quantity" like height or pH. And in the final part of the text, involving system whose configuration changes in time, we'll use "variable" in the sense of "a quantity that varies over time."

2.4 Computing: commands and evaluation

Mathematical notation is effective for **describing** functions and operations, but ***computing notation*** provides a way to go beyond the description to actually **carry out** the operations. Computer notation will be an equal partner to mathematical notation in *MOSAIC Calculus*.

With computers, writing an expression in computer notation goes hand-in-hand with ***evaluating*** the notation. We'll start with the simplest mode of evaluation, where you are writing the expression in the ***console*** for the language. Figure @ref(fig:R-console) shows and example the console tab provided by the RStudio application.

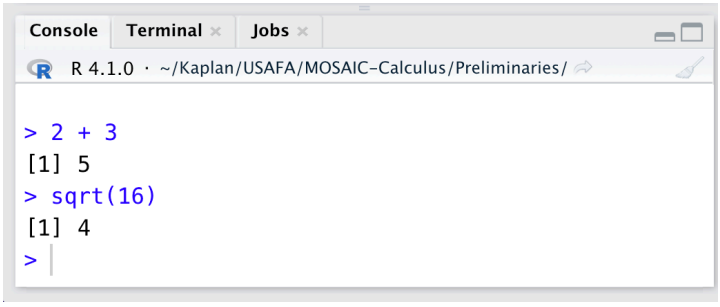


Figure 2.1: An RStudio console tab for writing expressions and evaluating them. The > is the ***prompt*** after which you write your expression, here shown in blue. Pressing the "return" key causes the language interpreter to evaluate the command.

In Figure 2.1 we have come in to the story in the middle of the action. To start, there was just a prompt character.

```
>
```

The person at the keyboard then typed a simple expression: `2 + 3`

```
> 2 + 3
```

Having completed the expression, the keyboarder presses "return." The RStudio application sends the expression to the software that "interprets" it according to the rules of the R language. Since `2 + 3` is a complete, valid R expression, the R-language software carries out the action specified—adding 2 and 3—and returns the result to RStudio, which displays it just below the expression itself.

```
> 2 + 3
[1] 5
```

Note that the value of the expression is simply the number `5`. The R language is set up to **format** numbers with an index, which is helpful when the value of the expressions is a large set of numbers. In the case here, with just a single number in the result of evaluating the expression, the index is simply stating the obvious.

Having printed the result of evaluating the `2 + 3` expression, RStudio shows another prompt, signally that it's ready for you to enter your next expression. In Figure 2.1) we're seeing the console after the person at the keyboard has responded to the prompt by writing another expression, pressed return, had RStudio print the value of that expression, and displayed a new prompt.

The two expressions shown in the console in Figure 2.1 both evaluate to single numbers. We say, "the command returns a value." The ***command*** is a valid R expression followed by the instruction ("Return") to evaluate the command. The ***value*** of the expression is the result of evaluating the command.

Another common form of R expression is called ***assignment***. An assignment means "giving a name to a value." It's done with a more complicated expression, like this:

```
b <- 22/7
```

The result of evaluating this command is to store in the computer memory, under the name `b`, the value of `22/7`. Because the value is being stored, R is designed *not* to display the value as happened with the first two commands in the console. If you want to see the value printed out, give the name as a command:

```
b
## [1] 3.142857
```

TAKE NOTE!

This book displays the command being evaluated in a gray box, without a prompt. The value returned by the command is displayed underneath the command, prefaced by ##. In the book formatting, the four commands we have just described would be displayed in this way:

```
2 + 3
## [1] 5
sqrt(16)
## [1] 4
b <- 22/7
b
## [1] 3.142857
```

When reading this book, take care to distinguish between the display of a command and the display of the value returned by that command. The first is something you type, the second is printed by the computer.

2.5 Functions in R/mosaic

One of the fundamental mathematical operations in this book is ***defining functions***. You've already seen the way we use mathematical notation to define a function, for instance,

$$h(t) \equiv 1.5\,t^2 - 2\ .$$

The R/mosaic equivalent to the definition of $h()$ is:

```
h <- makeFun(1.5*t^2 - 2 ~ t)
```

Once you have defined a function, you can evaluate it on an input. The R notation for evaluating functions is exactly the same as with mathematical notation, for instance,

```
h(4)
## [1] 22
```

or

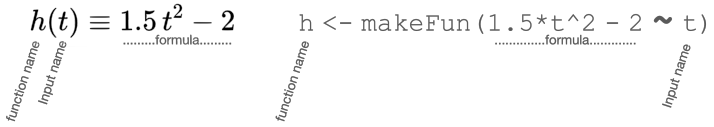
```
h(t=4)
## [1] 22
```

There are obvious differences, however, between the mathematical and computing notation used to define a function. All the same information is being provided, but the format is different. That information is:

1. the name of the function: $h()$ or `h`. When writing the name of a computer-defined function, we'll put the reminder parentheses after the name, as in `h()`.
2. the name of the input to the function: x or `x`
3. the calculation that the function performs, written in terms of the input name. $1.5t^2 - 2$ or `1.5 * t^2 - 2`.

Laying out the two notation forms side by side let's us label the elements they share:

For the human reading the mathematical notation, you know that the statement defines a function because you have been told so. Likewise, the computer needs to be told what to do



with the provided information. That's the point of `makeFun()`. There are other R/mosaic commands that could take the same information and do something else with it, for example create a graph of the function or (for those who have had some calculus) create the derivative or the anti-derivative of the function.

TAKE NOTE!

In R, things like `makeFun()` are called "functions" because, like mathematical functions, they turn inputs into outputs. In the case of `makeFun()`, the input is a form called a ***tilde expression***, owing to the character tilde (~) in the middle. On the right-hand side of the tilde goes the name of the input. On the left-hand side is the R expression for the formula to be used, written as always in terms of the input name. The whole tilde expression is taken as the one argument to `makeFun()`. Although it may seem odd to have punctuation in the middle of an argument, remember that something similar happens when we write $h(t = 3)$.

2.6 Names and assignment

The command

```
h <- makeFun(1.5*t^2 - 2 ~ t)
```

gives the name `h` to the function created by `makeFun()`. Good choice of names makes your commands much easier for the human reader.

The R language puts some restrictions on the names that are allowed. Keep these in mind as you create R names in your future work:

1. A name is the **only**[2] thing allowed on the left side of the assignment symbol `<-`.
2. A name must *begin* with a letter of the alphabet, e.g. `able`, `Baker`, and so on.
3. Numerals can be used after the initial letter, as in `final4` or `g20`. You can also use the period `.` and underscore `_` as in `third_place`. No other characters can be used in names: no minus sign, no `@` sign, no `/` or `+`, no quotation marks, and so on.

For instance, while `third_place` is a perfectly legitimate name in R, the following are not: `3rd_place`, `third-place`. But it's OK to have names like `place_3rd` or `place3`, etc., which start with a letter.

R also distinguishes between letter case. For example, `Henry` is a different name than `henry`, even though they look the same to a human reader.

2.7 Formulas in R

The constraint of the keyboard means that computer formulas are written in a slightly different way than the traditional mathematical notation. This is most evident when writing multiplication and exponentiation. Multiplication must *always* be indicated with the `*` symbol, for instance 3π is written `3*pi`. For exponentiation, instead of using superscripts like 2^3 you use the "caret" character, as in `2^3`. The best way to learn to implement mathematical formulas in a computer language is to read examples and practice writing them.

Here are some examples:

[2]Note for R experts: Strictly speaking, the thing to the left of `<-` must be an "assignable," which includes names with indices (e.g. `Engines$hp` or `Engines$hp[3:5]` and other forms). We will not use indexing in *MOSAIC Calculus*; we won't need it.

Traditional notation	R notation
$3 + 2$	`3 + 2`
$3 \div 2$	`3 / 2`
6×4	`6 * 4`
$\sqrt{4}$	`sqrt(4)`
$\ln 5$	`log(5)`
2π	`2 * pi`
$\frac{1}{2}17$	`(1 / 2) * 17`
$17 - 5 \div 2$	`17 - 5 / 2`
$\frac{17-5}{2}$	`(17 - 5) / 2`
3^2	`3^2`
e^{-2}	`exp(-2)`

Each of these examples has been written using numbers as inputs to the mathematical operations. The syntax will be exactly the same when using an input name such as `x` or `y` or `altitude`, for instance `(x - y) / 2`. In order for that command using `x` and `y` to work, some meaning must have been previously attached to the symbols. We'll come back to this important topic on another day.

2.8 Exercises

Exercise 2.1: TKWEW unassigned

Exercise 2.2: LDNE unassigned

Exercise 2.3: kZG5Fj unassigned

Exercise 2.4: aeOnO5 unassigned

Exercise 2.5: ooJK5d unassigned

Exercise 2.6: BXCA4 unassigned

Exercise 2.7: 0V510o R formula notation

Exercise 2.8: Ce79t3 makeFun()

Exercise 2.9: BaEJkS unassigned

2.9 Drill questions

3 Graphs and graphics

Before we go much further in exploring the creation and uses of functions, let's remember the general idea of mathematical modeling: the construction of mathematical representations of systems. The word "system" is familiar in everyday speech and is used to describe all manner of things: means of communication, ecology, politics, the workings of the market, etc. A "system" involves a group of related components that operates as a whole. For instance, the digestive system consists of body organs and reflexes that, collectively, transform food into the elementary substances needed for metabolism. In the economic theory of the market, components are prices, demand, and supply. These three components are not independent. Demand is related to price as is supply, both are what economists sometimes call "curves" but which we would call functions.

Key steps in making a mathematical representation—a model—of a system involve identifying the system components and describing quantitatively the relationships among them. In brief, mathematical modeling is about describing the ***relationships*** between things.

See how it's done.

Let's start with a simple system that is so familiar that you likely do not think of it as a system: a triangle.

A triangle consists of three connected line segments: the sides. It has other properties that are related to the sides and each other, for example, the angles between sides, the perimeter, or the area enclosed by the triangle.

Here's a description of the relationship between the perimeter p and the lengths of the sides, a, b, c, written in the form of a function:

$$p(a, b, c) \equiv a + b + c \ .$$

The mathematical expression of the relationship between area A enclosed by the triangle and the side lengths goes back at least 2000 years to Heron of Alexandria (circa 10–70). As a function, it can be written

$$A(a, b, c) \equiv \frac{1}{4}\sqrt{4a^2b^2 - (a^2 + b^2 - c^2)^2} \ .$$

FOR INSTANCE ...

Solid CO_2

There's considerable interest in ways to remove CO_2 from the atmosphere and store it permanently underground. It's hard to store gasses in the massive quantities needed to mitigate climate change. But CO_2_ storage is part of a system that includes temperature, pressure, and chemical affinity.

Figure 3.1 shows the relationship between physical form of pure CO_2_ and the temperature and pressure of the gas.

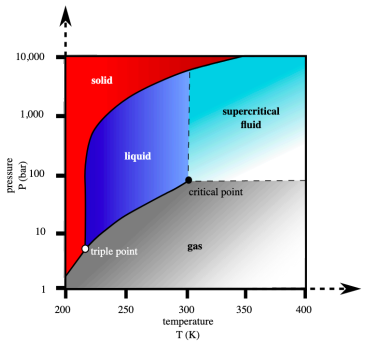


Figure 3.1: Phase diagram for CO_2_. Source Ben Finney, Mark Jacobs, CC0, via Wikimedia Commons

---

MATH IN THE WORLD ...

Nerve cells communicate via electrical voltages and currents. For long-distance communications (distances longer than about 1 mm) the signaling takes the form of pulses of voltage occurring repetitively at different rates. The formation of these pulses, called "action potentials," was the subject of an extensive research project in the 1950s involving inserting tiny electrodes

into relatively large ("giant") nerve cells that mediate the fleeing reaction of squid. A typical experiment involved regulating artificially the voltage across the cell membrane. By these experiments, the scientists— John Eccles, Alan Hodgkin and Andrew Huxley—were able to describe mathematically the relationship between membrane voltage and the current across the membrane. A calculus model built from the relationships provided a concise description of the biophysics of action potentials. A 1963 Nobel Prize was awarded for this work.

In each individual experimental run, the membrane voltage was fixed at a given level (say, -50 mV) and the current measured. Figure Figure 3.2 shows what the data might have looked like, each point showing the results of one experimental run.

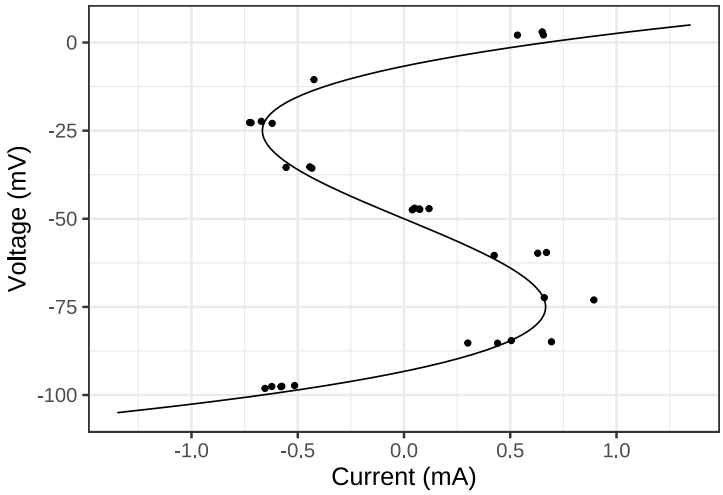


Figure 3.2: Data (simulated) from the squid giant axon experiments. A smooth curve is drawn through the data points.

The data points themselves might be described metaphorically as "clouds" spotting the voltage vs current "sky." Real-world clouds often show patterns such as animal or country shapes. We might say that a given cloud resembles a rabbit. Similarly, the data clouds show a pattern between current and voltage. We might, for instance, describe the current-voltage relationship as S-shaped. Or, rather than using the letter "S" we could draw a curve through the dots to summarize and simplify the relationship.

The smooth curve in Figure 3.2 describes the relationship be-

tween current and voltage quantitatively. For example, if you know that the current is 0, you can use the curve to figure out what the voltage will be around -90 mV or -50 mV or -10 mV. But when current is 0, the voltage will *not* be, say, -75 or -150.

Graphs such as Figure 3.1 and Figure 3.2 are good ways of showing relationships. We can even do calculations simply using such graphs. Place your finger on a point of the S-shaped graph and you can read from the axes an allowed pair of voltage and current values. Place your finger on a point on the vertical axis. Moving it to the curve will reveal what current is associated with that voltage.

Functions are, like graphs, a ways of representing relationships. For all their advantages as a means of communication, graphs have their limits. With a graph it's feasible only to show the relationship between two quantities or among three quantities. Functions, can involve more quantities. For instance, the triangle-area function

$$A(a,b,c) \equiv \frac{1}{4}\sqrt{4a^2b^2 - (a^2+b^2-c^2)^2}$$

gives the relationship between four quantities: the area and the lengths of the triangle's three sides.

On the other hand, functions cannot represent all types of relationships. For instance, the curve in Figure 3.2 shows a relationship between current and voltage in nerve cells. But there is no mathematical function voltage(current) that does justice to the relationship. The reason is that mathematical functions can have ***one and only one*** output for any given input. There are three reasonable values for membrane voltage that are experimentally consistent with a zero current, not just one.

Care needs to be taken in using functions to represent relationships. For the nerve-cell current-voltage relationship, for instance, we can construct a function current(voltage) to represent the relationship. That's because for any given value of voltage there is just one corresponding current. But there is no voltage(current) function, even though knowing the current tells you a lot about the voltage.

3.1 Function graphs

Given a function, it's easy to draw the corresponding relationship as a graphic. This section describes how to do that for functions that have one or two inputs. The opposite—given a relationship, represent it using functions—is not always so easy and will require modeling techniques that we'll develop in Block 1.

Contemporary practice is to draw graphs of functions using a computer. R/mosaic provides several functions that do this, you need only learn how to use them.

There are two essential arguments shared by all of the R/mosaic functions drawing a graph:

1. The function that is to be graphed. This is to be written as a ***tilde expression*** in exactly the same manner as described in Chapter (**notation-and-computing?**).
2. The ***domain interval***. The domain of many functions reaches to infinity, but our computer screens are not so big! Making a graph requires choosing a finite interval for each of the input variables.

The tilde expression for a function with one input will have only one name on the right-hand side of the ~. The domain interval specification must use the same name:

Tilde expression	Domain interval specification
`x^2 ~ x`	`interval(x = -3:3)`
`y * exp(y) ~ y`	`interval(y = 0:10)`
`log(y) / exp(y) ~ y`	`interval(y = -5:5)`
`sin(z) / z ~ z`	`interval(y = -3*pi:3*pi)`

3.1.1 Slice plot

To draw a graph of a function with one input, use `slice_plot()`. The tilde expression is the first argument; the domain interval specification is the second argument. For instance,

```
slice_plot(y * exp(y) ~ y, domain(y=0:10))
```

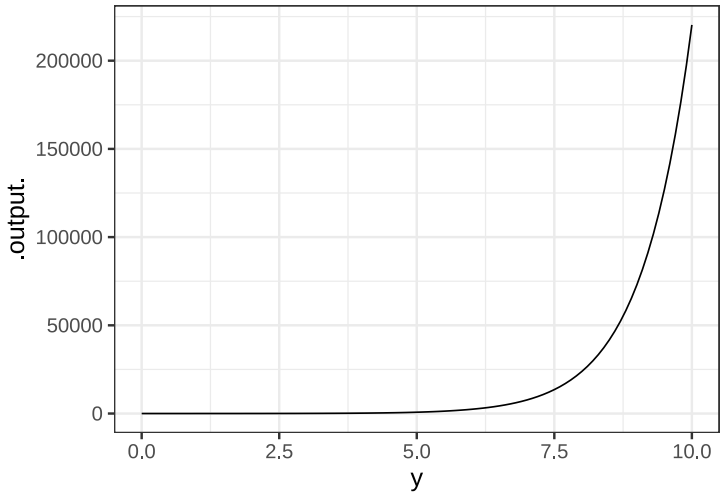


Figure 3.3: Graph of the function $f(y) \equiv ye^y$.

Recall the situation seen in Figure 3.2 which shows a two-dimensional space of all possible (voltage, current) pairs for nerve cells. The experimental data identified many possible pairs—marked by the dots in Figure Figure 3.2 —that are consistent with the relationships of the nerve-cell system.

The same is true of Figure 3.3, the graph of a function with a single input. The two-dimensional space shown in the Figure 3.3 contains (input, output) pairs, only a small fraction of which are consistent with the relationship described by the function. The points in that small fraction could be marked by individual dots, but instead of dots we draw the continuous curve connecting the dots. Every point on the curve is consistent with the relationship between input and output represented by the function.

The ***graphics frame*** in Figure 3.3 is a 2-dimensional area for drawing. The ***domain*** of the function being graphed in that frame, $f() \equiv ye^y$, is the number line, that is, the space of all real numbers. The plot, however, shows only a finite interval $0 \leq y \leq 10$ of that domain.

3.1.2 Contour plot

Functions with **two inputs** can be displayed with `contour_plot()`. Naturally, the tilde expression defining the function will have **two** names on the right-hand side of ~. Similarly, the domain specification will have two arguments, one for each of the names in the tilde expression.

```
contour_plot(exp(-z)*sin(y) ~ y & z, domain(y=-6:6, z=0:2))
```

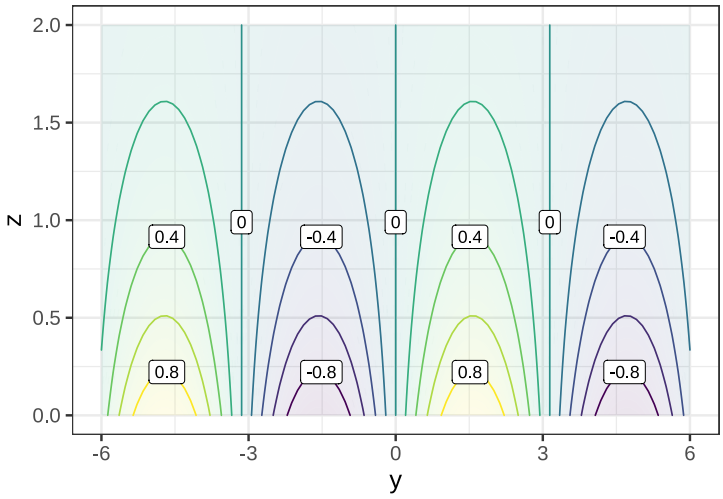


Figure 3.4: Contour plot of a function with two inputs $g(y, z) \equiv e^{-z} \sin(y)$

Contour plots will be a preferred format for displaying functions with two inputs. The main reason to prefer contour plots is the ease with which locations of points in the input space can be identified and the ability to read output values without much difficulty.

3.1.3 Surface plot

There's another way to think about graphing functions with two inputs. There are in such a situation **three** quantities involved in the relationship. Two of these are the inputs, the third is the output. A three-dimensional space consists of all the possible triples of point; the relationship between the inputs and the output is marked by ruling out almost all of the potential triples and marking those points in the space that are consistent with the function.

the space of all possibilities (y, z, output) is three-dimensional, but very few of those possibilities are consistent with the function to be graphed. You can imagine our putting dots at all of those consistent-with-the-function points, or our drawing lots and lots of continuous curves through those dots, but really the

cloud of dots forms a ***surface***; a continuous cloud of points floating over the (y, z) input space.

Figure 3.5 displays this surface. Since the image is drawn on a two-dimensional screen, we have to use painters' techniques of perspective and shading. In the interactive version of the plot, you can move the viewpoint for the image which gives many people a more solid understanding of the surface being shown.

```
surface_plot(exp(-z)*sin(y) ~ y & z, interval(y=-6:6, z=0:2))
```

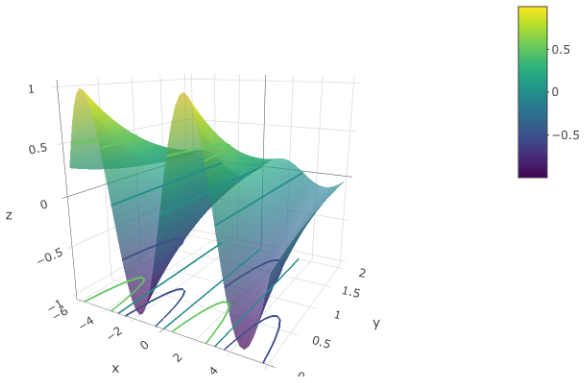


Figure 3.5: Displaying $g(y, z) \equiv e^{-z} \sin(y)$ as a surface plot annotated with contour lines.

Note that the surface plot is made with the R/mosaic `surface_plot()`, which takes arguments in exactly the same way as `contour_plot()`.

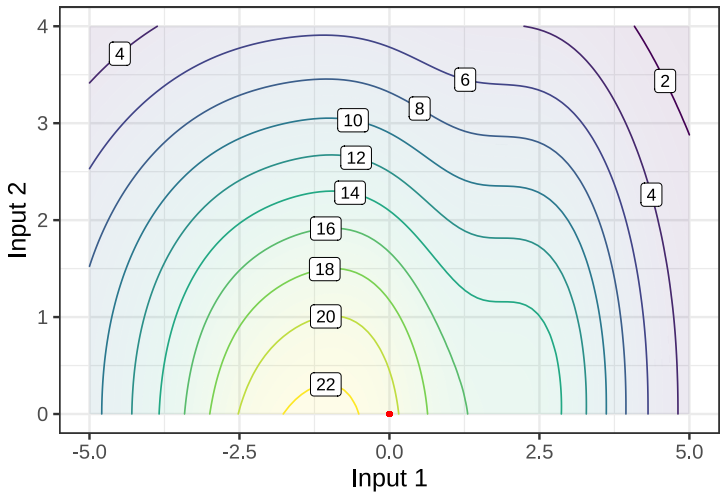
3.2 Interpreting contour plots

It may take some practice to learn to read contour plots fluently but it is a skill that's worthwhile to have. Note that the graphics frame is the Cartesian space of the two inputs. The output is presented as ***contour lines***. Each contour line has a label giving the numerical value of the function output. Each of the

input value pairs on a given contour line corresponds to an output at the level labeling that contour line. To find the output for an input pair that is *not* on a contour line, you ***interpolate*** between the contours on either side of that point.

See how it's done.

What's the value of the function being plotted here at input $(\text{input}_1 = 0, \text{input}_2 = 0)$?



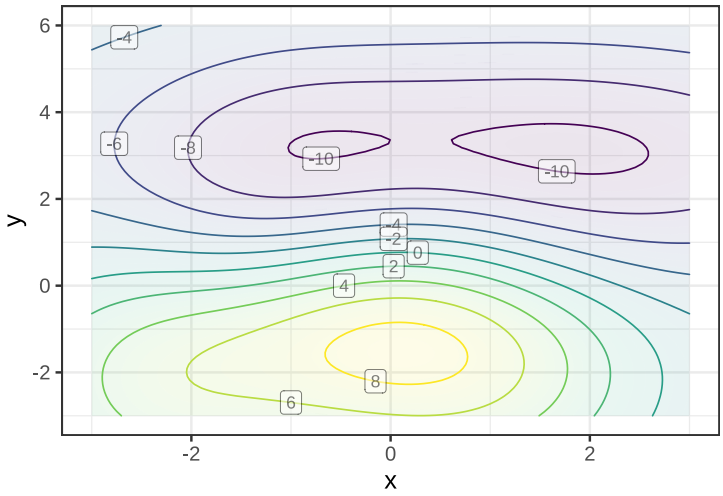
The input pair (0, 0)—which is marked by a small red dot—falls between the contours labeled "20" and "22." This means that the output corresponding to input (0, 0) is somewhere between 20 and 22. The point is much closer to the contour labeled "20", so it's reasonable to see the output value as 20.5. This is, of course, an approximation, but that's the nature of reading numbers off of graphs.

Often, the specific numerical value at a point is not of primary interest. Instead, we may be interested in how steep the function is at a point, which is indicated by the spacing between contours. When contours are closely spaced, the hillside is steep. Where contours are far apart, the hillside is not steep, perhaps even flat.

Another common task for interpreting contour plots is to locate the input pair that's at a local high point or low point: the top of a hill or the bottom of a hollow. Such points are called ***local argmax*** or ***local argmin*** respectively. The *output* of the function at a local argmax is called the ***local maximum***; similarly for a local argmin, where the output is called a ***local minimum***. (The word "argmax" is a contraction of "argument of the maximum." We will tend to use the word "input" instead of "argument", but it means exactly the same thing to say "the inputs to a function" as to says "the arguments of a function.")

See how it's done.

For this contour graph



i. Find an input coordinate where the function is steepest.
ii. Find input coordinates for the high and low points in the function .

A function is steepest where contour lines are spaced closely together, that is, where the function output changes a lot with a small change in input. This is true near inputs $(x = 0, y = 1)$. But notice that steepness involves a *direction*. Near $(x = 0, y = 1)$, changing the x value does not lead to a big change in output,

but a small change in the value of y leads to a big change in output. In other words, the function is steep in the y-direction but not in the x-direction.

The highest output value explicitly marked in the graph is 8. We can imagine from the shapes of the contours surrounding the 8 contour that the function reaches a peak somewhere in the middle of the region enclosed by the 8 contour, near the input coordinate $(x = 0, y = -1.5)$.

Similarly, the lowest output value marked is -10. In the middle of the area enclosed by the -10 contour is a local low point. That there are two such regions, one centered near input coordinate $(x = -0.5, y = 3.3)$, the other at $(x = 1.5, y = 3.1)$.

Why did you?

Why do you call the graphs of functions of one variables **slice plots** rather than simply graphs?

Saying "graph" for a display of $f(x)$ versus x is correct and reasonable. But in *MOSAIC Calculus* we have another point to make.

Almost always, when mathematically modeling a real-world situation or phenomenon, we do not try to capture every nuance of every relationship that might exist in the real world. We leave some things out. Such simplifications make modeling problems easier to deal with and encourage us to identify the most important features of the most important relationships.

In this spirit, it's useful always to assume that our models are leaving something out and that a more complete model involves a function with more inputs than the present candidate. The present candidate model should be considered as a ***slice*** of a more complete model. Our slice leaves out one or more of the variables in a more complete model.

To illustrate this, suppose that the actual system involves relationships among three quantitiess, which we represent in the

As you become practiced reading contour plots, you might prefer to read this one as a hilltop (shaded yellow) side-by-side with a hollow or bowl (shaded purple), with green, almost level flanks at the left and right edges of the frame.

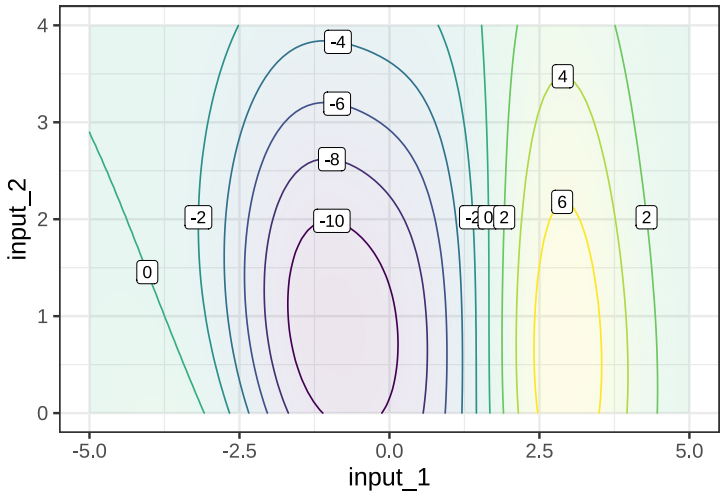


Figure 3.6: A hypothetical relationship among three quantities.

form of a function of two inputs, as shown in Figure 3.6. (The third quantity in the relationship is the output of the function.)

The most common forms of ***slice*** involve constructing a simpler function that has one input but not the other. For example, our simpler function might ignore input 22. There are different ways of collapsing the function of two inputs into a function of one input. An especially useful way in calculus is to take the two-input function and set one of the inputs to a ***constant value***.

For instance, suppose we set input 22 to the constant value 1.5. This means that we can consider any value of input 1, but input 2 has been replaced by 1.5. In Figure Figure 3.7, we've marked in red the points in the contour plot that give the output of the simplified function.

Each point along the red line corresponds to a specific value of input #1. From the contours, we can read the output corresponding to each of those values of input #1. This relationship, output versus input #1 can be drawn as a mathematical graph (to the right of the contour plot). Study that graph until you can see how the rising and falling parts of the graph correspond to the contours being crossed by the red line.

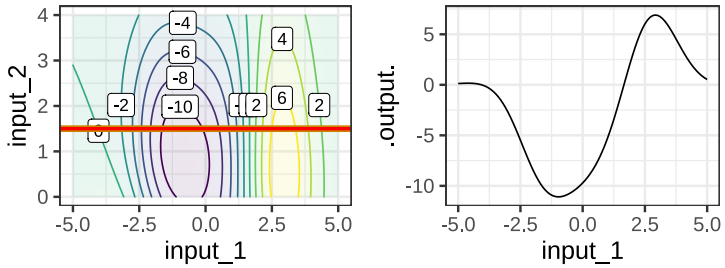


Figure 3.7: Left: A slice (red) through the domain of a contour plot. Right: The value of the function along the red slice presented as a mathematical graph, generated by `slice_plot()`.

Slices can be taken in any direction or even along a curved path! The blue line in Figure 3.8 shows the slice constructed by letting input 2 vary and holding input 1 at the constant value 0.

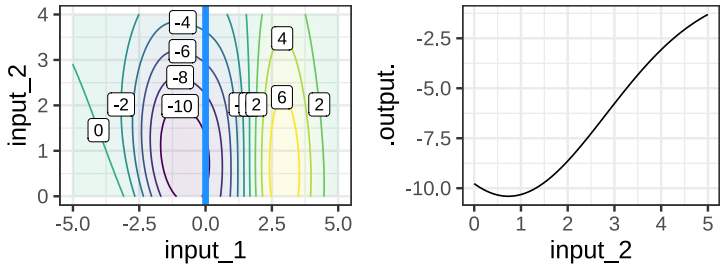


Figure 3.8: A path (blue) along which to cut a slice. The graph is made with `contour_plot()`. Right: Slice plot along the blue path. The graph is made with `slice_plot()`.

3.3 Exercises

Exercise XX.XX: sR6PVw reading slice plots

Exercise XX.XX: bYHEy6 read surface plots

EXERCISE: USE a plot like `surface_plot(exp(-z)*sin(y) ~ y & z, interval(y=-6:6, z=0:2), type="contour")` so that students can read off the (x,y,z) value. Ask for the value of the function at several input points. Ask them to trace the output value as the cursor moves along a contour line.

EXERCISE: Here are some additional tasks which you should learn to perform at a glance when reading a contour plot:

1. Start at a given input pair and determine two directions:

    a. the direction to move which is most steeply uphill,
    b. the direction to move which will keep the function output the same.

2. Translate a small region of a contour plot into the word for a corresponding geographical feature with that topology: hills, valleys, crests, coves, hollows, and so on.

EXERCISE: Match up the slice plots with the paths indicated on the contour plot.

EXERCISE: Ask them to find the actual lowest point in the graph

4 Pattern-book functions

In this Chapter, we introduce the ***pattern-book functions***–a short list of simple mathematical functions that provide a large majority of the tools for representing the real world as a mathematical object. Think of the items in this list as different actors, each of whom is skilled in portraying an archetypal character: hero, outlaw, lover, fool, comic. A play brings together different characters, costumes them, and relates them to one another through dialog or other means. All this is for the purpose of telling a story and providing insight into human relationships and emotions.

A mathematical model is a kind of story and a mathematical modeler a kind of playwright. She combines mathematical character types to tell a story about relationships. We need only a handful of mathematical functions, the analog of the character actors in drama and comedy in order to rough-out a model. In creating a mathematical model, you clothe the actors to suit the era and location and assemble them together in harmony or discord.

Costume designers and others do not start from nothing. There are reference guides that collect patterns which a designer can customize to the needs at hand. These reference guides are sometimes called "pattern books." (See Figure 4.1.)

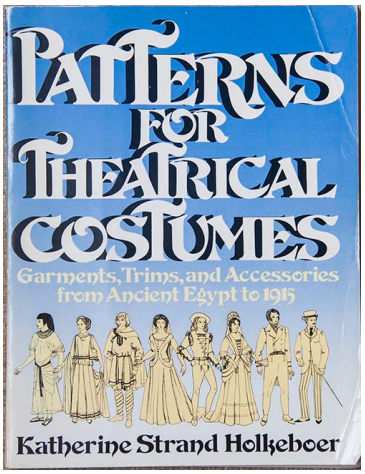


Figure 4.1: A pattern book of theatrical costumes

Similarly, we will start with a pattern set of functions that have been collected from generations of experience. To remind us of their role in modeling, we'll call these ***pattern-book functions***. These pattern-book functions are useful in describing diverse aspects of the real world and have simple calculus-related properties that make them relatively easy to deal with. There are just a handful pattern-book functions from which untold numbers of useful modeling functions can be constructed. Mastering calculus is in part a matter of becoming familiar with the

mathematical connections among the pattern-book functions. (You'll see these in due time.)

Here is a list of our pattern-book functions showing both a traditional and the R notation:

The list of pattern-book functions is short. You should memorize the names and be able easily to associate each name with the traditional notation.

Pattern name	Traditional notation	R notation
exponential	e^x	`exp(x)`
logarithm ("natural log")	$\ln(x)$	`log(x)`
sinusoid	$\sin(x)$	`sin(x)`
square	x^2	`x^2`
proportional	x	`x`
one	1	`1`
reciprocal	$1/x$ or x^{-1}	`1/x`
gaussian	$\text{dnorm}(x)$	`dnorm(x)`
sigmoid	$\text{pnorm}(x)$	`pnorm(x)`

These functions are shown with a traditional formula notation to highlight the connections to the math you already studied, and x is used as the input to these functions out of tradition.

Over the next several chapters, we will introduce several features of functions. These features include:

- monotonicity up or down
- concavity up or down
- horizontal asymptotes
- vertical asymptotes
- periodicity
- continuity

By the end of Chapter (**describing-functions?**), you should be able to list all the basic pattern-book functions and describe the features relevant to each.

These pattern-book functions are widely applicable. But nobody would confuse the pictures in a pattern book with costumes that are ready for wear. Each pattern must be tailored to fit the actor and customized to fit the theme, setting, and action of the story. We'll study how this is done starting in Chapter (**parameters?**).

WHY DID YOU?

There is universal agreement about the names of all of the pattern-book functions except for two: the gaussian and the logarithm.

The name "gaussian" is not descriptive; the graph of a gaussian function looks like a camel's hump, the curve of a bell, or a bump in the road. There are all sorts of bell-shaped functions that differ slightly in their origins and detailed shape. In this book, we may use the terms "hump", "bell", or "bump" to remind you of the basic shape, but the specific mathematical function we have in mind is called the ***gaussian function***, named after a tremendously influential mathematician, Carl Friedrich Gauss (1777-1855). This function, first published in 1718 (before Gauss was born), has an important role throughout physical science, technology, and data science. In probability theory and the social sciences, the shape of the function is given the simple name "normal distribution"; it shows up in so many places to be considered utterly unsurprising and "normal."

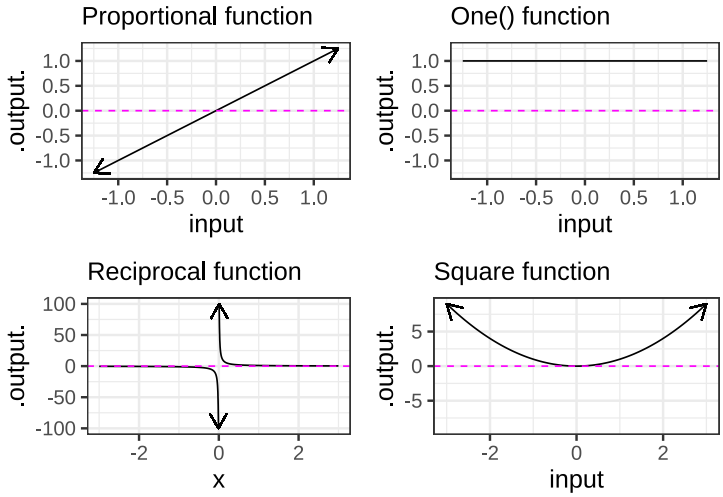
The name "logarithm" is anything but descriptive. The name was coined by the inventor, John Napier (1550-1617), to emphasize the original purpose of his invention: to simplify the work of multiplication and exponentiation. The name comes from the Greek words *logos*, meaning "reasoning" or "reckoning," and *arithmos*, meaning "number." A catchy marketing term for the new invention, at least for those who speak Greek!

Although invented for the practical work of numerical calculation, the logarithm function has become central to mathematical theory as well as modern disciplines such as thermodynamics and information theory. The logarithm is key to the measurement of information and magnitude. As you know, there are units of information used particularly to describe the information storage capacity of computers: bits, bytes, megabytes, gigabytes, and so on. Very much in the way that there are different units for length (cm, meter, kilometer, inch, mile, …), there are different units for information and magnitude. For almost everything that is measured, we speak of the "units" of measurement. For logarithms, instead of "units" by tradition another word is used: the ***base*** of the logarithm. The most common units outside of theoretical mathematics are

base-2 ("***bit***") and base-10 ("***decade***"). But the unit that is most convenient in mathematical notation is "base e," where $e = 2.71828182845905....$ This is genuinely a good choice for the units of the logarithm, but that's hardly obvious to anyone encountering it for the first time. To make the choice more palatable, it's marketed as the "base of the natural logarithm." In this book, we'll be using this ***natural logarithm*** as our official pattern-book logarithm.

4.1 Function shapes

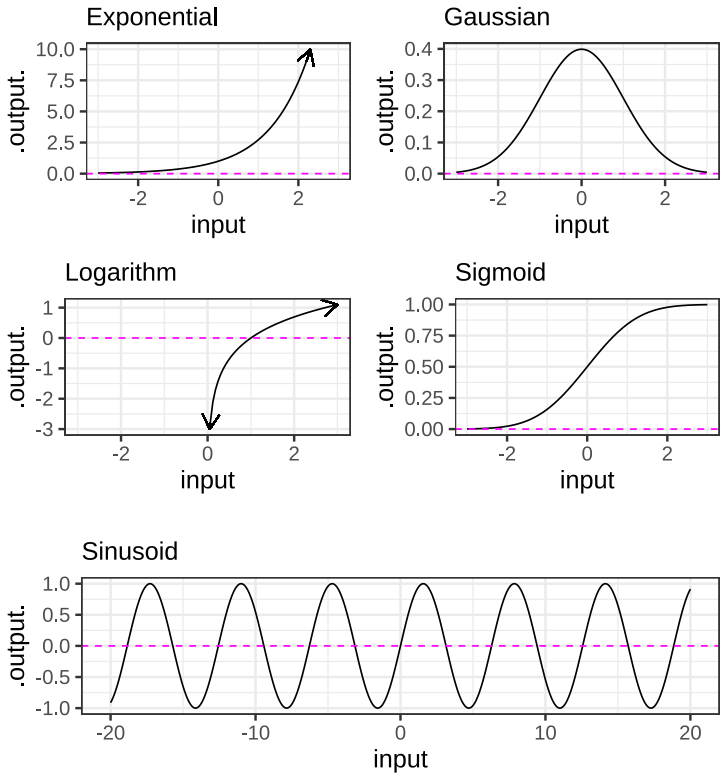
You are going to be building models by selecting an appropriate function or by putting functions together in various ways. This might remind you of Lego blocks. As you know, Legos come in different shapes: 6×2, 4×2, 2×2, and so on. Similarly, each of the pattern-book functions has a distinctively shaped graph. Knowing the shapes by name will help you when you need to build a model.



The ***proportional function*** and the ***constant function*** have extremely simple shapes. Note that the graph of a constant function is not just any line, but a line with zero slope.

It is tempting to deny that the constant function is a function. After all, the output does not depend on the input. Still, this situation arises frequently in modeling: you start out supposing that one quantity depends on another, but it sometimes turns out that it does not. Since functions are our way of representing relationships, it is helpful to have a function for the situation of "no relationship." The constant function does this job.

You may also notice the proportional function shown above does nothing to change the input; it simply returns as output the same value it received as input. For this reason, the proportional pattern-book function ($f(x) = x$) is sometimes called the ***identity function***. Later we will show how this simple pattern-book function can be endowed with parameters that form the basis of many change relationships.



WHY DID YOU?

How come some function names, like sin() are written with other parentheses, while others such as e^x have the input name shown?

The x in the name e^x is really a placeholder. A better, if longer-winded name would be exp(), which would signal that we mean the abstract concept of the exponential function, and not that function applied to an input named x.

The same is true for functions like x or $1/t$ or z^2. If absolute consistency of notation were the prime goal, we could have written this book in a style that gives a name to every pattern-book function in the name/parentheses style. Something like this:

```
reciprocal <- makeFun(1/t ~ t)
one <- makeFun(1 ~ z)
square <- makeFun(x^2 ~ x)
```

These would be used in the ordinary way, for instance:

```
reciprocal(7)
## [1] 0.1428571
one(123.67)
## [1] 1
square(19)
## [1] 361
```

Writing reciprocal(x) instead of $1/x$ is long-winded, which is perhaps why you never see it. But when you see $1/x$ you should think of it as a function being applied to x and not as a bit of arithmetic.

By the way … I used different names for the inputs in these three functions just to remind the reader that, for functions with one input, the name has no significance. You just have to make sure to use the same name on the left- and right-hand sides of the tilde expression.

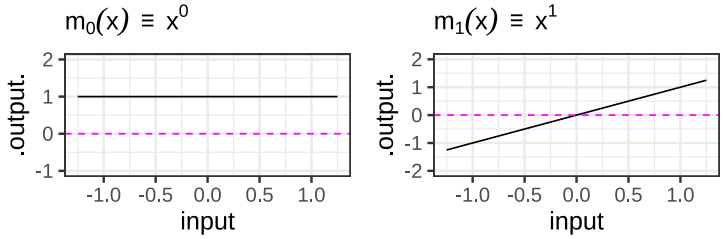
4.2 The power-law family

Three of the pattern-book functions—$1/x$, x, x^2—actually belong to an infinite family called the ***power-law functions***. The three shown above occur so often and are so closely related to the other pattern-book functions that they receive their own special names (inverse, proportional, and square) and place in our list; however, it is the *family* of power-law functions that form the more general pattern needed for modeling.

Some other examples of power-law functions are $x^3, x^4, ...$ as well as $x^{1/2}$ (also written $\sqrt{x}$), $x^{1.36}$, and so on. Some of these also have special (albeit less frequently used) names, but *all* of the power-law functions can be written as x^p, where x is the input and p is a number. .

Within the power-law family, it is helpful to know and be able to distinguish between several overlapping groups:

1. The ***monomials***. These are power-law functions such as $m_0(x) \equiv x^0$, $m_1(x) \equiv x^1$, $m_2(x) \equiv x^2$, ..., $m_p(x) \equiv x^p$, ..., where p is a whole number (i.e., a non-negative integer). Of course, $m_0()$ is exactly the same as the constant function, since $x^0 = 1$. Likewise, $m_1(x)$ is the same as the identity function since $x^1 = x$. As for the rest, they have just two general shapes: both arms up for even powers of p (like in x^2, a parabola); one arm up and the other down for odd powers of p (like in x^3, a cubic).



2. The ***negative powers***. These are power-law functions where $p < 0$, such as $f(x) \equiv x^{-1}$, $g(x) \equiv x^{-2}$, $h(x) \equiv x^{-1.5}$. For negative powers, the size of the output is ***inversely proportional*** to the size of the input. In other words, when the input is large (**not** close to zero) the

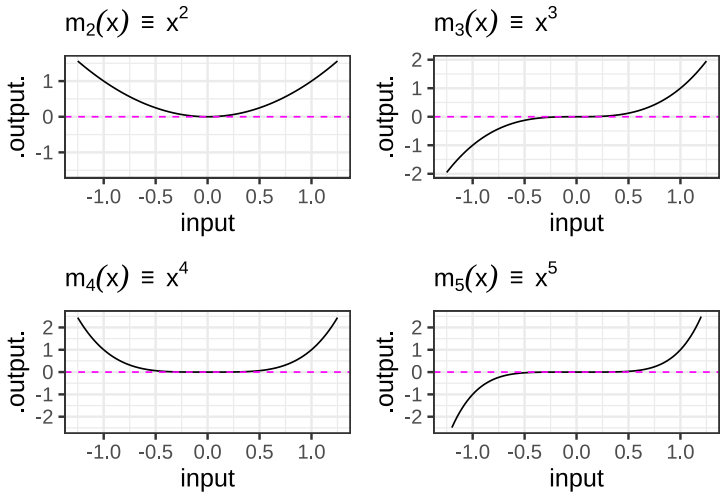


Figure 4.2: Graphs of the monomial functions from order 0 to 5. The dashed magenta line marks zero output.

output is small, and when the input is small (close to zero), the output is *very* large. This behavior happens because a negative exponent like x^{-2} can be rewritten as $\frac{1}{x^2}$; the input is *inverted* and becomes the denominator, hence the term "inversely proportional".

3. The **non-integer powers**, e.g. $f(x) = \sqrt{x}$, $g(x) = x^{\pi}$, and so on. When p is either a fraction or an irrational number (like π), the real-valued power-function x^p can only take non-negative numbers as input. In other words, the domain of x^p is 0 to ∞ when p is not an integer. You have likely already encountered this domain restriction when using the power law with $p = \frac{1}{2}$, since $f(x) \equiv x^{1/2} = \sqrt{x}$, and the square root of a negative number is not a *real* number. You may have heard about the *imaginary* numbers that allow you to take the square root of a negative number, but for the moment, you only need to understand that when working with real-valued power-law functions with non-integer exponents, the input must be non-negative. (The story is actually a bit more complicated since, algebraically, rational exponents like 1/3 or 1/5 with an odd-valued denominator can be applied to negative numbers. Computer arithmetic, however, does

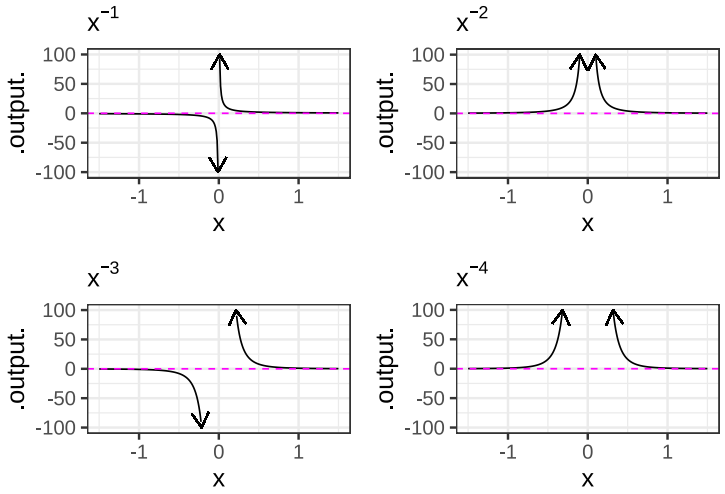


Figure 4.3: Graphs of power-law functions with negative integer exponents. The arrows point to the output being very large when x is near zero.

not recognize these exceptions.)

4.3 Domains of pattern-book functions

Each of our basic modeling functions, with two exceptions, has a domain that is the entire number line $-\infty < x < \infty$. No matter how big or small is the value of the input, the function has an output. Such functions are particularly nice to work with, since we never have to worry about the input going out of bounds.

The two exceptions are:

1. the logarithm function, which is defined only for $0 < x$.
2. some of the power-law functions: x^p.

    - When p is negative, the output of the function is undefined when $x = 0$. You can see why with a simple example: $g(x) \equiv x^{-2}$. Most students had it drilled into them that "division by zero is illegal," and $g(0) = \frac{1}{0}\frac{1}{0}$, a double law breaker.

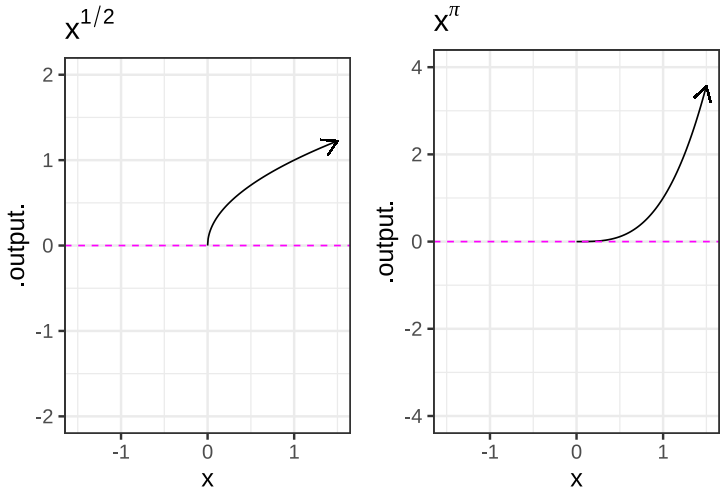


Figure 4.4: The domain of power-law functions with non-integer power is $0 \leq x < \infty$.

- When p is not an integer, that is $p \neq 1, 2, 3, \cdots$ the domain of the power-law function does not include negative inputs. To see why, consider the function $h(x) \equiv x^{1/3}$.

It can be tedious to make sure that you are on the right side of the law when dealing with functions whose domain is not the whole number line. The designers of the hardware that does computer arithmetic, after several decades of work, found a clever system to make it easier. It's a standard part of such hardware that whenever a function is handed an input that is not part of that function's domain, one of two special "numbers" is returned. To illustrate:

```
sqrt(-3)
## [1] NaN
(-2)^0.9999
## [1] NaN
1/0
## [1] Inf
```

`NaN` stands for "not a number." Just about any calculation involving `NaN` will generate `NaN` as a result, even those involving

multiplication by zero or cancellation by subtraction or division.[1] For instance:

```
0 * NaN
## [1] NaN
NaN - NaN
## [1] NaN
NaN / NaN
## [1] NaN
```

Division by zero produces `Inf`, whose name is reminiscent of "infinity." `Inf` infiltrates any calculation in which it takes part:

```
3 * Inf
## [1] Inf
sqrt(Inf)
## [1] Inf
0 * Inf
## [1] NaN
Inf + Inf
## [1] Inf
Inf - Inf
## [1] NaN
1/Inf
## [1] 0
```

To see the benefits of the `NaN` / `Inf` system let's plot out the logarithm function over the graphics domain $-5 \leq x \leq 5$. Of course, part of that graphics domain, $-5 \leq x \leq 0$ is not in the domain of the logarithm function and the computer is entitled to give us a slap on the wrists. The `NaN` provides some room for politeness.

Open an R console see what happens when you make the plot.

```
library(Zcalc)
slice_plot(log(x) ~ x, interval(x=c(-5,5)))
```

[1] One that does produce a number is `NaN^0`.

4.4 Symbolic manipulations

Several of the pattern book functions appear so often in *MOSAIC Calculus* that it's worth reviewing how to manipulate them symbolically. As an example, consider the function

$$g(x) \equiv e^x e^x \ .$$

This is a perfectly good way of defining $g()$, but it's helpful to be able to recognize that the following definitions are exactly equivalent

$$f(x) \equiv e^{x+x} h(x) \equiv e^{2x} \ .$$

The ***symbolic manipulation*** we touch on in this chapter involves being able to recall and apply such equivalences. We'll need only a small set of them, all or most of which are found in a high-school algebra course.

TAKE NOTE!

We'll be working with exponential and with power-law functions in this chapter. It is essential that you recognize that these are utterly different functions.

An exponential function, for instance, e^x or 2^x or 10^x has a constant quantity raised to a power set by the input to the function.

A power-law function works the reverse way: the input is raised to a constant quantity, as in x^2 or x^10.

A mnemonic phrase for exponentials functions is

Exponential functions have x in the exponent.

Of course, exponential functions can has inputs with names other than x, for instance $f(y) \equiv 2^y$, but the name "x" makes for a nice alliteration in the mnemonic.

4.4.1 Exponential and logarithm

We will Basic symbolic patterns for exponentials are

$$\textbf{(i)} \qquad e^x e^y \leftrightarrow e^{x+y}$$

$$\textbf{(ii)} \qquad \left(e^x\right)^y \leftrightarrow e^{x\,y}$$

Exponentials with a base other than e can be converted to base e.

$$\textbf{(iii)} \quad 2^x \leftrightarrow e^{\ln(2)x} = e^{0.69315x} \qquad \textbf{(iv)} \quad 10^x \leftrightarrow e^{\ln(10)x} = e^{2.30259x}$$

It's usually easier for people to deal with bases 2 and 10 because they can do the multiplications in their head. The base $e = 2.718282$ is not conducive to mental arithmetic. In Block 2 we'll discuss why it's standard to write an exponential function as e^x.

The logarithms, which we'll return to in Chapter 15 are the ***inverse*** function of the exponential. That is:

$$\textbf{(v)} \qquad \ln(\exp(x)) = x \quad \text{and conversely} \quad \exp(\ln(x)) = x\ .$$

One place that we will encounter rules (ii) and (v) is in Chapter 8 when we look a "doubling times" and "half lives." There we will deal with expressions such as $2^{x/\tau} = e^{\ln(2)x/\tau}$.

Important symbolic patterns for logarithms are

$$\textbf{(vi)} \quad \ln(x\ y) \leftrightarrow \ln(x)+\ln(y) \qquad \textbf{(vii)} \quad \ln(x/y) \leftrightarrow \ln(x)-\ln(y) \qquad \textbf{(viii)} \quad \ln(x^p) \leftrightarrow p\ln(x)$$

TAKE NOTE!

Notice that the symbolic patterns for logarithms involve multiplication, division, and exponentiation, but **not addition**: $\ln(x+y) \neq \ln(x) + \ln(y)$.

4.4.2 Power-law functions

In power-law functions, the quantity in the exponent is constant: we'll call it m, n, or p in the following examples.

(**ix**) $x^m x^n \leftrightarrow x^{m+n}$(**x**) $\frac{x^m}{x^n} \leftrightarrow x^{m-n}$(**xi**) $\left(x^n\right)^p \leftrightarrow x^{n\,p}$(**xii**) $x^{-n} \leftrightarrow \frac{1}{x^n}$ (**xiii**) $x^0 \leftrightarrow 1$

4.4.3 Sinusoids

$\sin(x)$ is periodic with period 2π. Zero-crossings of $\sin(x)$ are at $x = ..., -2\pi, -\pi, 0, \pi, 2\pi, ...$

$\cos(x) = \sin\left(x + \frac{\pi}{2}\right)$

There's a sine and a cosine. They are shifted versions of one another.

Phase shift.

Linear combination to represent phase shift. $\sin(x + \phi) = \cos(\phi)\sin(x) + \sin(\phi)\cos(x)$

$C\sin(x + \phi) = A\sin(x) + B\cos(x)$ where $C^2 = A^2 + B^2$ and $\phi = \arctan(A/B)$

TAKE NOTE!

When a function like $\sqrt[3]{x}$ is written as $x^{1/3}$ make sure to include the exponent in grouping parentheses: `x^(1/3)`. Similarly, later in the course you will encounter power-law functions where the exponent is written as a formula. Particularly common will be power-law functions written x^{n-1} or x^{n+1}. In translating this to computer notation, make sure to put the formula within grouping parentheses, for instance `x^(n-1)` or `x^(n+1)`.

4.5 The straight-line function

You are probably used to a function that we call the "straight-line function"

$$\text{line}(x) \equiv mx + b \ .$$

The name comes from the shape of a graph of line(x) versus x, which is a straight line. An you are likely used to calling the parameter m the "slope" of the line, and the parameter b the "intercept." (In general, by "intercept" we will mean the value of the function output when the input is zero. In high-school, this is often called the "y-intercept.")

There are two simple symbolic manipulations that you will be using often in *MOSAIC Calculus*:

1. Find the input $x = x_0$ for which the output line($x = x_0$) $= 0$. This input has many names in mathematics: the "root," the "x-intercept," the "zero crossing," etc. We will call any input value that corresponds to an output of zero to be "a zero of the function."

For the straight-line function, the zero is readily found symbolically:

$$x_0 = -b/m \ .$$

2. Re-write the straight-line function in the form

$$\text{line}(x) = a\,(x - x_0) \ .$$

Here, the slope is designated with a. And, of course, x_0 is the zero of the function, as you can see by setting $x = x_0$: $\text{line}(x = x_0) = a(x - x_0)\big|_{x=x_0} = 0$.

4.6 Functions with multiple inputs

Each of our pattern-book function has a single input. In most applications

4.7 Exercises

Exercise 4.1: H2KG3 unassigned

Exercise 4.2: Pdt9jy unassigned

Exercise 4.3: VIW7T unassigned

Exercise 4.4: ILESX unassigned

Exercise 4.5: Qj0JAr unassigned

Exercise 4.6: DLWSA unassigned

Exercise 4.7: RLUCX unassigned

Exercise 4.8: MNCLS2 unassigned

EXERCISES: Show that $\ln(x^p) = p\ln(x)$

4.8 Drill

5 Describing functions

Knowing and correctly using a handful of phrases about functions with a single input goes a long way in being able to communicate with other people . Often, the words make sense in everyday speech ("steep", "growing", "decaying", "goes up", "goes down", "flat").

Sometimes the words are used in everyday speech but the casual person isn't sure exactly what they mean. For instance, you will often hear the phrase "growing exponentially." The graph of the exponential function illustrates exactly this sort of growth: flat for small x and growing steadily steeper and steeper as x increases.

Still other words are best understood by those who learn calculus. "Concave up," "concave down", "approaching 0 asymptotically," "continuous", "discontinuous", "smooth", "having a minimum **at** ...," "having a minimum **of** ...", "approaching ∞ asymptotically," "having a vertical asymptote."

The next short sections describe seven simple function-shape concepts: slope, concavity, continuity, monotonicity, periodicity, asymptotes, and local extrema. Each of these concepts has the idea of a function at the core, because each one depends on how the function output changes as the input is changed.

I'll illustrate these concepts using three pattern-book functions graphed in Figure 5.1.

5.1 Slope

The slope describes whether the output goes up or down, and to what extent, as the input changes. Typically, the slope is different for different input values. The only function type that

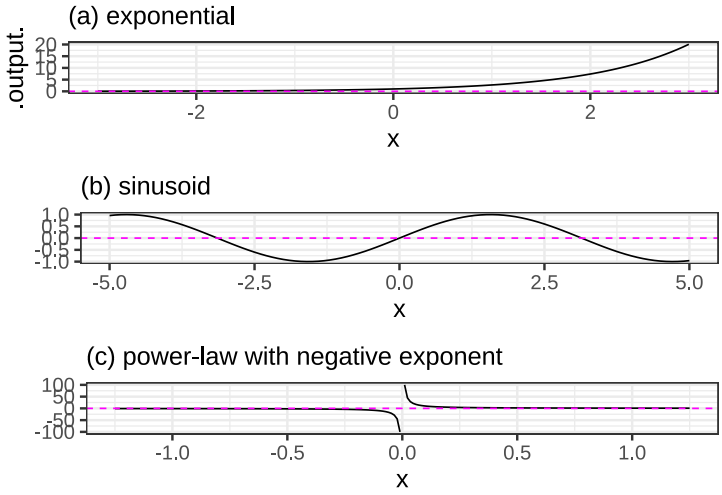


Figure 5.1: Three of the pattern-book functions: (a) exponential, (b) sinusoid, (c) power-law x^{-1}.

has a slope that is the same for all inputs is the straight-line function.

Figure 5.2 graphs the sinusoid function (black curve). At numerous points in the domain, the function has been overlaid with a straight-line segment that has the same slope as does the function itself. For x near -3 the slope is negative; for x near zero the slope is positive, then swings back to negative again for x near 3.

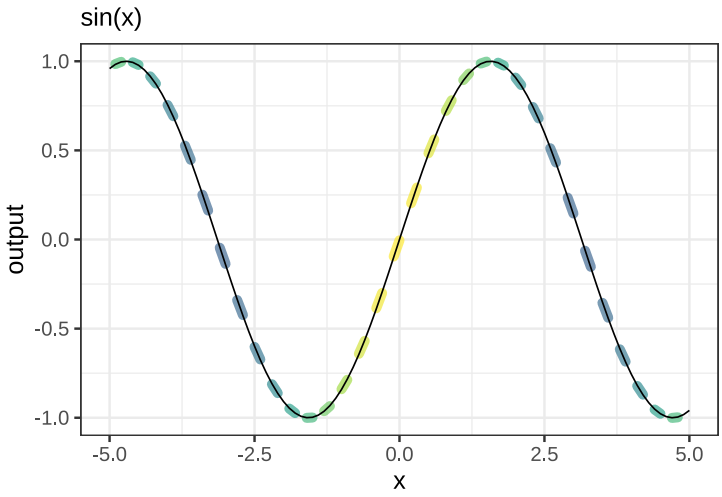


Figure 5.2: Short straight-line segments laid over a graph of the sinusoid. The slope of each line segment is selected to match the local slope of the sinusoid.

When we speak of the slope of the sinusoid, or any other function, we mean the local slope as a function of the input. The value of the function doesn't enter into it, just the slope. Fig-

ure 5.3 shows only the slope of the sinusoid, without the sinusoid output at all. Each line segment has a horizontal "run" of 0.1, so you can measure the slope of each segment—rise over run—as the vertical extent Δy of the segment divided by 0.1.

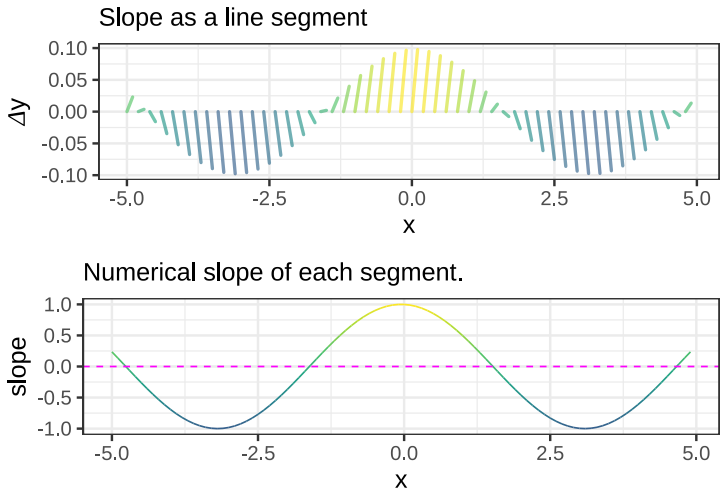


Figure 5.3: Showing just the slope of the sinusoid as a function of input x. Top: representing the slope by the steepness of line segments. Bottom: The numerical value of the slope of each segment.

For instance, the Δy for the slope segment at $x = 0$ is 0.1, so the slope at $x = 0$ is $\Delta y/0.1 = 1$. At $x = 1$, $\Delta y \approx 0.05$, so the slope is 0.5. The graph colors the segment according to the slope, so large negative slopes are blue, slopes near zero are green, and large positive slopes are yellow.

TAKE NOTE!

A more general word than "slope" for describing functions is ***rate of change***. It's absolutely crucial to distinguish between the ***change*** in the output value of a function and the ***rate of change*** of that output.

To illustrate, suppose we have a function $f(x) \equiv x^2 + 3$. When we talk about "change" we imagine a situation where we have to different values of the function ***input***, say $x_1 = 3$ and $x_2 = 6$.

The "change" in output for these two different inputs is $f(x_2) - f(x_1)$, or in this case $39 - 12 = 27$.

In contrast, the "rate of change" is the change in output **divided by** the change in input, that is:

$$\frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{27}{3} = 9 \ .$$

A "rate" in mathematics is a ratio: one measure divided by another. For instance, a heart rate is measured as beats-per-minute. To measure it, count the number of pulse waves in a given interval of time. A typical medical practice is to count for 15 seconds, an interval long enough to get a reliable count but short enough not to unduly prolong the process. If 18 pulse waves were counted in the 15 seconds, the heart rate is 18 beats per 15 seconds, more usually reported as 72 beats-per-minute.

In a rate of change, the ratio is the change in output divided by the change of input.

5.2 Concavity

The slope of a function at a given input tells how fast the function **output** is increasing or decreasing as the input changes slightly. ***Concavity*** is not directly about how the function output changes, but about how the function's *slope* changes. For instance, a function might be growing slowly in some region of the domain and then gradually shift to larger growth in an adjacent region. Or, a function might be decaying steeply and then gradually shift to a slower decay. Both of these are instances of ***positive concavity***. The opposite pattern of change in slope is called ***negative concavity***. If the slope doesn't change at all—only straight-line functions are this way— the concavity is zero.

Concavity has a very clear appearance in a function graph. If a function is positive concave in a region, the graph looks like a smile or cup. Negative concavity looks like a frown. Zero concavity is a straight line.

Referring to the three function examples in Figure 5.1), we'll use the traditional terms ***concave up*** and ***concave down*** to refer to positive and negative concavity respectively.

a. The exponential is ***concave up*** everywhere in its domain.
b. The sinusoid alternates back and forth between ***concave up*** and ***concave down***.
c. This particular power law x^{-1} is ***concave up*** for $0 < x$ and ***concave down*** for $x < 0$.

When a function switches between positive concavity and negative concavity, as does the sinusoid as well as the gaussian and sigmoid functions, there is an input value where the switch occurs and the function has zero concavity. (Continuous functions that pass from negative to positive or *vice versa* must always cross zero.) Such in-between points of zero concavity are called ***inflection points***. A function can have zero, one, or many inflection points. For instance, the sinusoid has inflection points at $x = \ldots, -\pi, 0, \pi, 2\pi, \ldots$, the cubic power function $f(x) \equiv x^3$ has one, and the exponential has none.

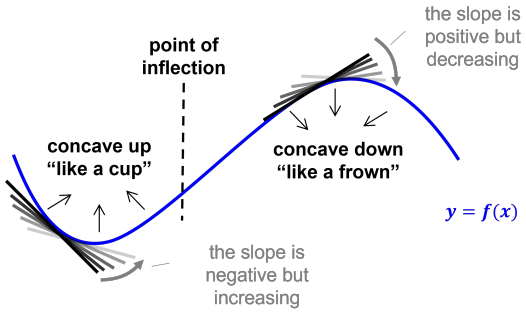


Figure 5.4: A cubic function which is concave up until a point of inflection and concave down thereafter. [Source: Maj. Austin Davis]

"Inflection point" appears in news stories, so it is important to know what it means in context. The mathematical definition is about the change in the direction of curvature of a graph. In business, however, it generally means something less esoteric, "a time of significant change in a situation" or "a turning point."[1] The business sense effectively means that the function—say profits as a function of time, or unemployment as a function of time—has a non-zero concavity, up or down. It's about the *existence* of concavity rather than about the change in the sign of concavity.

One of the benefits of learning calculus is to gain a way to think about the previous paragraph that's systematic, so it's

[1] Google dictionary, provided by Oxford Languages

always easy to know whether you are talking about the slope of a function or the ***change in slope*** of a function.

5.3 Continuity

A function is ***continuous*** if you can trace out the graph of the function without lifting pencil from the page. A function is ***continuous on an interval (a,b)*** if you can trace the function over that whole interval. All of the pattern-book functions are continuous over any interval in their domain except for power-law functions with negative exponents. (This includes the reciprocal since it is a power-law with a negative exponent: $1/x = x^{-1}$.) Those exceptions are not defined at $x = 0$.

To illustrate, consider the power-law function graphed in Figure 5.1. On any interval (a,b) that does **not include 0**, the function is continuous. For inputs $x < 0$, the function is negative. For inputs $0 < x$, the function is positive. So, on an interval that includes $x = 0$ the function jumps discontinuously from negative to positive.

5.4 Monotonicity

A function is ***monotonic*** on a domain when the *sign* of the slope never changes on that domain. Monotonic functions either steadily **increase** in value or, alternatively, steadily **decrease** in value.

Another way of thinking about monotonicity is to consider the order of inputs and outputs compared to a number line. If a function is monotonically increasing then it will preserve the order of inputs along the number line when it maps inputs to outputs, whereas a monotonically decreasing function will reverse the order. For instance, if the input x comes before an input y (i.e., $x < y$), then $f(x) < f(y)$ for monotonically *increasing* functions (the order is preserved), but $f(y) < f(x)$ for monotonically *decreasing* functions (the order of outputs is reversed).

Of the pattern-book functions in Figure 5.1: both the exponential and the logarithm function are monotonic: the exponential grows monotonically as does the logarithm. The sinusoid is not monotonic over any domain longer than half a cycle: the function switchs between positive slope and negative slope in different parts of the cycle.

5.5 Periodicity

A phenomenon is ***periodic*** if it repeats a pattern over and over again. The pattern that is repeated is called a **cycle**; the periodic function as a whole is one cycle placed next to the previous one and so forth. The day-night cycle is an example of a periodic phenomenon, as is the march of the seasons. The ***period*** is the duration of one complete cycle; the period of the day-night cycle is 24 hours, the period of the seasonal progression is 1 year.

Real-world periodic phenomena often show some slight variation from one cycle to the next. Of the pattern-book functions, only the sinusoid is periodic. And it is exactly periodic, repeating exactly the same cycle over and over again. The period—that is, the length of an input interval that contains exactly one cycle—has a value of 2π for the pattern-book sinusoid. When used to model a periodic phenomenon, the model function needs to be tailored to match the period of the phenomena.

The idea of representing with sinusoids phenomena that are almost but not exactly periodic, for instance a communications signal or a vibration, is fundamental to many areas of physics and engineering.

5.6 Asymptotic behavior

Asymptotic refers to two possible situations depending on whether the input *or* output is being considered:

1. When the **input** to a function gets bigger and bigger in size, going to ∞ or $-\infty$. If, as the input changes in this way the output gets closer and closer to a specific value, the function is said to have a ***horizontal asymptote*** of that value.

An example in Figure 5.1 is the exponential function. As $x \to -\infty$, that is, as x goes more and more to the left of the domain, the output tends asymptotically to zero.

2. When the **output** of a function gets bigger and bigger in size, going to ∞ or $-\infty$ without the input doing likewise. The visual appearance on a graph is like a sky-rocket: the output changes tremendously fast even though the input changes only a little. The vertical line that the skyrocket approaches is called a ***vertical asymptote***. The power-law function x^{-1} has a ***vertical asymptote*** at $x = 0$. If you were to consider inputs closer and closer to $x = 0$, the outputs would grow larger and larger is magnitude, tending toward ∞ or $-\infty$.

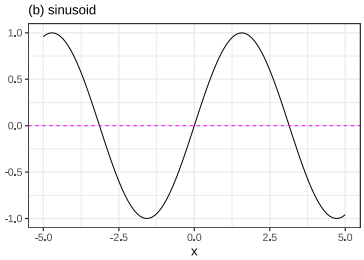
Several of the pattern-book functions have horizontal or vertical asymptotes or both. For instance, the function x^{-1} has a horizontal asymptote of zero for both $x \to \infty$ and $x \to -\infty$.

The sinusoid has neither a vertical nor a horizontal asymptote. As input x increases either to $-\infty$ or ∞, the output of the sinusoid continues to oscillate, never settling down to a single value. And, of course, the output of the sinusoid is everywhere $-1 \leq \sin(x) \leq 1$, so there is no possibility for a vertical asymptote.

5.7 Locally extreme points

Many continous functions have a region of the input domain where the output is gradually growing, then reaches a peak, then gradually diminishes. This is called a ***local maximum***. "Maximum" because the output reaches a peak at a particular input, "local" because in the neighborhood of the peak the function output is smaller than at the peak.

Likewise, functions can have a ***local minimum***: the bottom of a bowl rather than the top of a peak.



Of the three pattern-book functions in Figure 5.1, only the sinusoid has a local maximum, and, being periodic, it repeats that every cycle. The sinusoid similarly has a local minimum in every cycle..

Many modeling applications involve finding an input where the function output is maximized. Such an input is called an ***argmax***. "Argument" is a synonym for "input" in mathematical and computer functions, so "argmax" refers to the input at which the function reaches a maximum output. For instance, businesses attempt to set prices to maximize profit. At too low a price, sales are good but income is low. At too high a price, sales are too low to bring in much income. There's a sweet spot in the middle.

Other modeling applications involve finding an ***argmin***, the input for which the output is minimized. For instance, aircraft have a speed at which fuel consumption is at a minimum for the distance travelled. All other things being equal, it's best to operate at this speed.

The process of finding an argmin or an argmax is called ***optimization***. And since maxima and minima are very much the same mathematically, collectively they are called ***extrema***.

Any function that has an extremum cannot possibly be monotonic, since the growth is positive on one side of the extremum and negative on the other side.

5.8 Exercises

Exercise 4.1: H2eu2 unassigned

Exercise 4.2: PYKG5 unassigned

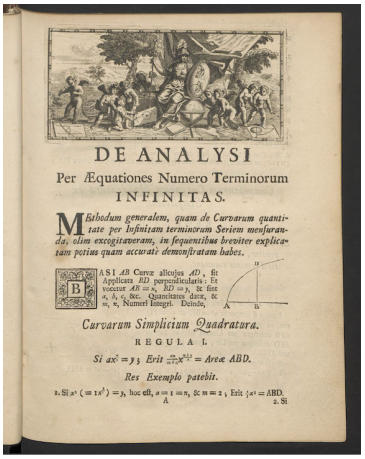
EXERCISE: Draw the slope at each of many points of these functions.

EXERCISE: Show the slope diagram for a few functions: as the students to match the slope diagram to the function.

5.9 Drill

6 Data and data graphics

The decade of the 1660s was hugely significant in the emergence of science, although no one realized it at the time. 1665 was a plague year, the last major outbreak of bubonic plague in England. The University of Cambridge closed to wait out the plague, and Isaac Newton, then a 24-year old Cambridge student returned home to Woolsthorpe where he lived and worked in isolation for two years. Biographer James Gleich wrote: "The plague year was his transfiguration. Solitary and almost incommunicado, he became the world's paramount mathematician." During his years of isolation, Newton developed what we now call "calculus" and, associated with that, his theory of Universal Gravitation. He wrote a tract on his work in 1669, but withheld it from publication until 1711.



DE ANALYSI

Per Æquationes Numero Terminorum INFINITAS.

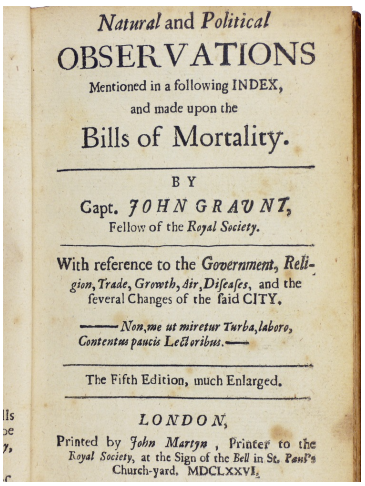
Curvarum Simplicium Quadratura.

REGULA I.

Res Exemplo patebit.

Figure 6.1: Newton's first tract (1669) on the roots of calculus: *On Analysis by Equations with an infinite number of terms.*

Plague was also the motivating factor in another important work, published in 1661, *Natural and Political Observations … Made upon the Bills of Mortality* by John Graunt (1620-1674). Bills of mortality, listings of the number and causes of deaths in London, had been published intermittently starting in in the plague year of 1532, and then continuously from the onset of plague in 1603. Graunt, a haberdasher by profession, performed what we might now call ***data science***, the extraction of information from data. For instance, Graunt was the first to observe the high rate of child mortality and that the number of deaths attributed to plague was underestimated by about one quarter. Graunt's work led to his election to the Royal Society, the same august group of scientists of which Isaac Newton was a member (and later president). Graunt is considered the first demographer and epidemiologist.



Natural and *Political*

OBSERVATIONS

Mentioned in a following INDEX,

and made upon the

Bills of Mortality.

BY

Capt. JOHN GRAUNT,

Fellow of the *Royal Society*.

With reference to the *Government, Religion, Trade, Growth, Air, Diseases*, and the several Changes of the said CITY.

——Non, me ut miretur Turba, laboro, Contentus paucis Lectoribus.——

The Fifth Edition, much Enlarged.

LONDON,

Printed by *John Martyn*, Printer to the *Royal Society*, at the Sign of the *Bell* in St. *Paul's* Church-yard, MDCLXXVI.

Figure 6.2: The title page of John Graunt's *Natural and Political Observations … upon the Bills of Mortality*

Graunt's publication marks the start of ***statistics***. He built upon a century of work by the city of London, collecting and tabulating data on a quarter of a million deaths. Indeed the

word “statistics” stems from “state,” the only entity large enough to collect data on populations and the economy.

Whereas advances in calculus over the next two centuries could be accomplished by the creativity of individuals, statistics could only develop based on the development of the infrastructure of government data collection, a process that took almost two centuries. But in the last 50 years, extensive data collection has entered non-state domains, such as genetic sequencing, remote sensing of Earth, commercial records, and the vast data warehouses of the social media giants, among others.

The ability to draw conclusions from masses of data—shown originally by John Graunt—is now an essential skill throughout science, government, and commerce. As you will see, particularly in Block 5, many of those skills are mathematical, effectively a part of calculus.

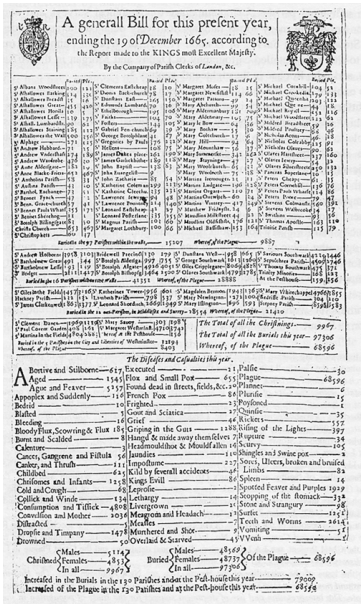
This chapter introduces some pre-calculus basics of working with data which we’ll use extensively in later Blocks of *MOSAIC Calculus*.

6.1 Data frames

Most people encounter data in the form of printed tables, such as the 1665 Bill of Mortality shown below. These tables were developed to be legible to humans and to be compact when printed.

Although published more than 350 years ago, it’s still possible for a literate human to sort out what the table is saying. But the volume of data has exploded beyond any possibility of putting it in print. Instead, today’s data are stored and accessed electronically. But the process of accessing such data is very much rooted in the notation of “table,” albeit tables that follow a strict set of principles. We’ll call such tables ***data frames*** and it’s important for you to learn a few of the core principles of data organization.

First, recognize that the data shown in Figure 6.3 consist of several different tables. The first table, just under the title, starts with


A generall Bill for this present year, ending the 19 of *December* 1665. according to the Report made to the KINGS most Excellent Majesty.

By the Company of Parish Clerks of *London*, &c.

Figure 6.3: Tables of data from the 1665 Bill of Mortality for London.

	Buried	Plague		Buried	Plague	
S^{t} Albans Woodstreet	100	121	S^{t} Clemens Eastcheap	18	20	···
S^{t} Alhallowes Barking	514	330	S^{t} Diones Back-church	78	27	···

The modern form of this is not spread out across the width of the page, and has an unique identifier for every column, as in

parish	buried	plague
S^{t} Albans Woodstreet	100	121
S^{t} Clemens Eastcheap	18	20
S^{t} Alhallowes Barking	514	330
S^{t} Diones Back-church	78	27
⋮	⋮	⋮

Each ***column*** of the modern table is called a ***variable***. So there is a variable "buried" that contains the number buried and another variable "plague" containing the number who died of plague.

Each row of the table is called a ***case***, but often simply ***row*** is used. For each table, all the cases are the same kind of thing, for instance, here, a parish.

Another table displayed on the sheet is entitled, "The diseases and casualities this year." In a modern format, this table starts out:

condition	deaths	year
Abortive and Stilborne	617	1665
Aged	1545	1665
Ague and Feaver	5257	1665
Appoplex and Suddenly	116	1665
Bedrid	10	1665

In this table, the case is a disease or other cause of death, which we've put under the name "condition." We've added the vari-

able "year" with an eye toward consolidating many years of the Bills into one table.

Toward the bottom of the sheet are entries that in a modern format would be two separate tables: one for people christened and one for people who died of plague.

Christened

males	females	year
5114	4853	1665

Plague

males	females	year
48569	48737	1665

In the Christened and Plague tables, the case is "one year." (We're imagining that the data from other years would go in additional rows.)

In modern data, summaries such as the "In all" for the Christened and Plague tables would not be included. Such summaries are easily computed as needed using appropriate database commands. Indeed, a contemporary organization assembling the dataset might organize the records into just two tables: Deaths and Births.

The Deaths table might look like:

name	date	parish	sex	age	cause
Percivell Bulling-ham	1665-06-01	S^{t} Mary le Bow	M	29	plague
Owin Swan-cott	1665-08-13	Trinitie	M	2	plague
Winifred Rom-ford	1665-11-09	S^{t} Swithings	F	19	childbed

name	date	parish	sex	age	cause
Elsebeth Cook	1665-06-29	St Ethelborough	F	5	plague
Humfray Langham	1665-06-05	St Bennet Fynch	M	53	aged
Agnes Kirkwood	1665-11-22	St Mary Hill	F	21	ague
Katherine Murton	1665-12-01	St Alholowes Lesse	F	24	childbed
Bainbridge Fletcher	1665-03-17	St Martins	M	2	plague
Cicely Ouston	1665-03-08	St Austins	F	35	plague

In the Deaths table, which would have 97,306 rows for 1665, the each case is "a person who died." Such a table could nowadays be re-tabulated into the "diseases and casualities" table, or the breakdown of burials by sex, or the parish-by-parish breakdown. But there are many other possibilities: looking at cause of death by age and season of year, or broken down by sex, etc.

6.2 Accessing data tables

In a data science course you will learn several ways of storing and accessing tables of data. One of the most important in professional use is a ***relational database***. ("Relation" is another word for "table," just as functions are about the relationship between inputs and output.)

Data wrangling is a term used to describe working with and summarizing data. This includes merging multiple data frames. In *MOSAIC Calculus* our uses of data will be focused on constructing functions that show the patterns in data and plotting data to reveal those patterns to the eye.

For our work, you can access the data frames we need directly in R by name. For instance, the `Engines` data frame records the characteristics of several internal combustion engines of various sizes:

Engine	mass	BHP	RPM	bore	stroke
Webra Speed 20	0.25	0.78	22000	16.5	16
Enya 60-4C	0.61	0.84	11800	24.0	22
Honda 450	34.00	43.00	8500	70.0	58
Jacobs R-775	229.00	225.00	2000	133.0	127
Daimler-Benz 609	1400.00	2450.00	2800	165.0	180
Daimler-Benz 613	1960.00	3120.00	2700	162.0	180
Nordberg	5260.00	3000.00	400	356.0	407
Cooper-Bessemer V-250	13500.00	7250.00	330	457.0	508

Various attributes of internal combustion engines, from the very small to the very large.

6.3 Variable names

The fundamental questions to ask first about any data frame are:

i. What constitutes a row?
ii. What are the variables and what do they stand for?

The answers to these questions, for the data frames we will be using, are available via R documentation. To bring up the documentation for `Engines`, for instance, give the command:

```
?Engines
```

When working with data, it's common to forget for a moment what are the variables, how they are spelled, and what sort of values each variable takes on. Two useful commands for reminding yourself are (illustrated here with `Engines`):

```
names(Engines) # the names of the variables
## [1] "Engine"       "mass"         "ncylinder"    "strokes"      "displacement"
## [6] "bore"         "stroke"       "BHP"          "RPM"
```

```
head(Engines) # the first several rows
## # A tibble: 6 x 9
##   Engine            mass ncylinder strokes displacement  bore stroke   BHP   RPM
##   <chr>            <dbl>     <dbl>   <dbl>        <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1 Webra Speedy     0.135         1       2          1.8  13.5   12.5  0.45 22000
## 2 Motori Cipolla   0.15          1       2          2.5  15     14    1    26000
## 3 Webra Speed 20   0.25          1       2          3.4  16.5   16    0.78 22000
## 4 Webra 40         0.27          1       2          6.5  21     19    0.96 15500
## 5 Webra 61 Blackh~ 0.43          1       2         10    24     22    1.55 14000
## 6 Webra 6WR        0.49          1       2         10    24     22    2.76 19000
nrow(Engines) # how many rows
## [1] 39
```

In RStudio, the command `View(Engines)` is useful for showing a complete table of data.

6.4 Plotting data

We will use just one graphical format for displaying data: the ***point plot***. In a point plot, also known as a "scatterplot," two variables are displayed, one on each graphical axis. Each case is presented as a dot, whose horizontal and vertical coordinates are the values of the variables for that case. For instance:

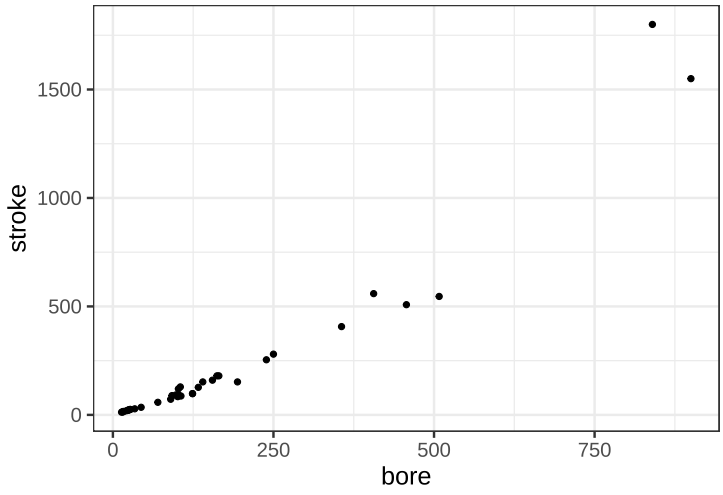


Figure 6.4: A point plot showing the relationship between engine `stroke` and `bore`. Each individual point is one row of the data frame SHOWN IN GIVE TABLE A NAME

The data plotted here show a relationship between the stroke length of a piston and the diameter of the cylinder in which the piston moves. This relationships, however, is not being presented in the form of a function, that is, a single stroke value for each value of the bore diameter.

For many modeling purposes, it's important to be able to represent a relationship as a function. At one level, this is straightforward: draw a smooth curve through the data and use that curve for the function.

Later in *MOSAIC Calculus*, we'll discuss ways to construct functions that are a good match to data using the pattern-book functions. Here, our concern is graphing such functions on top of a point plot. So, without explanation (until later chapters), we'll construct a power-law function, called, stroke(bore), that might be a good match to the data. The we'll add a second layer to the point-plot graphic: a slice-plot of the function we've constructed.

```
stroke <- fitModel(stroke ~ A*bore^b, data = Engines)
gf_point(stroke ~ bore, data = Engines) %>%
  slice_plot(stroke(bore) ~ bore, color="blue")
```

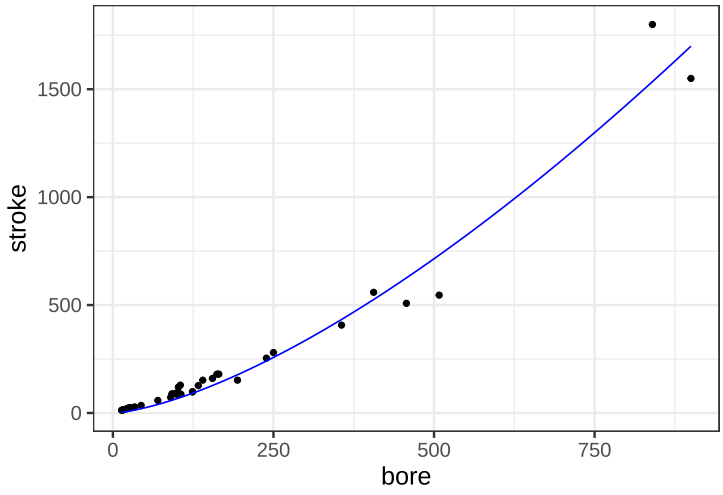
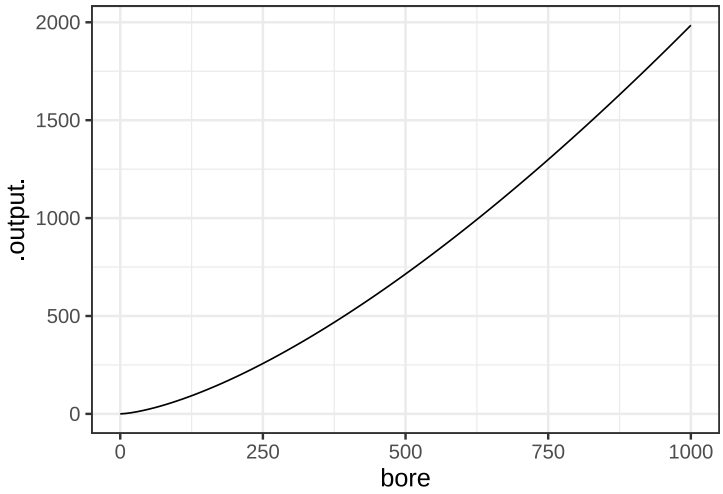


Figure 6.5: A graphic composed of two layers: 1) a point plot; 2) a slice plot of a function fitted to the data in (1).

The second layer is made with an ordinary `slice_plot()` command. To place it on top of the point plot we connect the two commands with a bit of punctuation called a "pipe": `%>%`.

`slice_plot()` is a bit clever when it is used after a previous graphics command. Usually, you need to specify the interval of the domain over which you want to display the function, as with



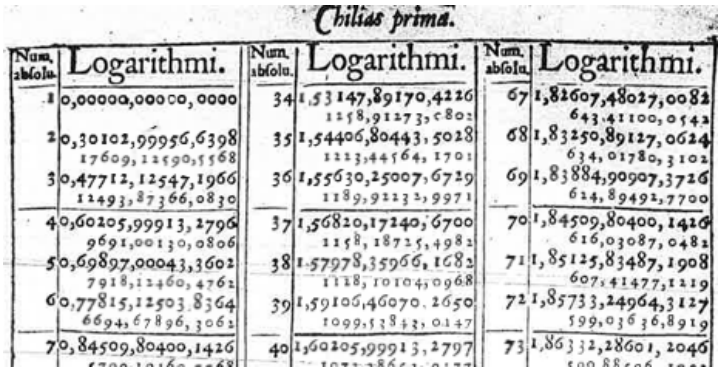
You can do that also when `slice_plot()` is the second layer in a graphics command. But `slice_plot()` can also infer the interval of the domain from previous layers.

The pipe punctuation can never go at the start of a line. Usually, we'll use the pipe at the very end of a line; think of the pipe as connecting one line to the next.

6.5 Functions as data

In the previous chapters, we've used ***formulas*** to define functions. The link between functions and formulas is important, but not at all essential to the idea of functions.

Arguably more important in practice to the representation of functions are ***tables*** and ***algorithms***. The computations behind the calculation of the output of functions such as sin() or e^x or other foundational functions that we'll introduce in Chapter @ref(pattern-book-functions) relies on computer software that loops and iterates and which is invisible to almost everybody who uses it. Before the advent of modern computing, functions were presented as printed tables. For instance, the logarithm function, invented about 1600, relied almost complete on printed tables, such as the one shown in Figure 6.6.



Chilias prima.

Num. absolu.	Logarithmi.	Num. absolu.	Logarithmi.	Num. absolu.	Logarithmi.
1	0,00000,00000,0000	34	1,53147,89170,4226 1258,91273,c802	67	1,82607,48027,0082 643,41100,0542
2	0,30102,99956,6398 17609,12590,5568	35	1,54406,80443,5028 1223,44564,1701	68	1,83250,89127,0624 634,01780,3102
3	0,47712,12547,1966 12493,87366,0830	36	1,55630,25007,6729 1189,92132,9971	69	1,83884,90907,3726 624,89492,7700
4	0,60205,99913,2796 9691,00130,0806	37	1,56820,17240,6700 1158,18725,4982	70	1,84509,80400,1426 616,03087,0482
5	0,69897,00043,3602 7918,12460,4762	38	1,57978,35966,1682 1128,10104,0968	71	1,85125,83487,1908 607,41477,1219
6	0,77815,12503,8364 6694,67896,3062	39	1,59106,46070,2650 1099,53843,0147	72	1,85733,24964,3127 599,03636,8919
7	0,84509,80400,1426	40	1,60205,99913,2797	73	1,86332,28601,2046

Figure 6.6: Part of the first table of logarithms, published by Henry Briggs in 1624.

In (**chap-pattern-book-functions?**) we'll introduce a small set of functions which we call the "pattern-book functions." Each of the functions is indeed a pattern that could be written down once and for all in tabular form. Generating such tables originally required the work of human "computers" who undertook extensive and elaborate arithmetical calculations by hand. What's considered the first programmable engine, a mechanical device designed by Charles Babbage (1791-1871) and programmed by Ada Lovelace (1815-1852), was conceived for the specific purpose of generating printed tables of functions.

It's helpful to think of functions, generally, as a sort of data storage and retrieval device that uses the input value to locate the corresponding output and return that output to the user. Any device capable of this, such as a table or graph with a human interpreter, is a suitable way of implementing a function.

To reinforce this idea, we ask you to imagine a long corridor with a sequence of offices, each identified by a room number. The input to the function is the room number. To ***evaluate*** the function for that input, you knock on the appropriate door and, in response, you'll receive a piece of paper with a number to take away with you. That number is the output of the function.

This will sound at first too simple to be true, but … In a mathematical function each office gives out exactly the same number every time someone knocks on the door. Obviously, being a worker in such an office is highly tedious and requires no special skill. Every time someone knocks on the worker's door, he

or she writes down the *same* number on a piece of paper and hands it to the person knocking. What that person will do with the number is of absolutely no concern to the office worker.

The utility of such functions depends on the artistry and insight of the person who creates them: the ***modeler***. An important point of this course is to teach you some of that artistry. Hopefully you will learn through that artistry to translate your insight to the creation of functions that are useful in your own work. But even if you just use functions created by others, knowing how functions are built will be helpful in using them properly.

In the sort of function just described, all the offices were along a single corridor. Such functions are said to have ***one input***, or, equivalently, to be "functions of one variable." To operate the function, you just need one number: the address of the office from which you'll collect the output.

Many functions have more than one input: two, three, four, … tens, hundreds, thousands, millions, …. In this course, we'll work mainly with functions of two inputs, but the skills you develop will be applicable to functions of more than two inputs.

What does a function of two inputs look like in our office analogy? Imagine that the office building has many parallel corridors, each with a numeric ID. To evaluate the function, you need two numeric inputs: the number of the corridor and the number of the door along that corridor. With those two numbers in hand, you locate the appropriate door, knock on it and receive the output number in return.

Three inputs? Think of a building with many floors, each floor having many parallel corridors, each corridor having many offices in sequence. Now you need three numbers to identify a particular office: floor, corridor, and door.

Four inputs? A street with many three-input functions along it. Five inputs? A city with many parallel four-input streets. And on and on.

Applying inputs to a function in order to receive an output is only a small part of most calculations. Calculations are usually organized as ***algorithms***, which is just to say that algorithms

are descriptions of a calculation. The calculation itself is … a function!

How does the calculation work? Think of it as a business. People come to your business with one or more inputs. You take the inputs and, following a carefully designed protocol, hand them out to your staff, perhaps duplicating some or doing some simple arithmetic with them to create a new number. Thus equipped with the relevant numbers, each member of staff goes off to evaluate a particular function with those numbers. (That is, the staff member goes to the appropriate street, building, floor, corridor, and door, returning with the number provided at that office.) The staff re-assembles at your roadside stand, you do some sorting out of the numbers they have returned with, again following a strict protocol. Perhaps you combine the new numbers with the ones you were originally given as inputs. In any event, you send your staff out with their new instructions—each person's instructions consist simply of a set of inputs which they head out to evaluate and return to you. At some point, perhaps after many such cycles, perhaps after just one, you are able to combine the numbers that you've assembled into a single result: a number that you return to the person who came to your business in the first place.

A calculation might involve just one function evaluation, or involve a chain of them that sends workers buzzing around the city and visiting other businesses that in turn activate their own staff who add to the urban tumult.

Take note!

The reader familiar with floors and corridors and office doors may note that the addresses are ***discrete***. That is, office 321 has offices 320 and 322 as neighbors. Calculus is about functions with a ***continuous domain***. But it's easy to create a continuous function out of a discrete table by adding on a small, standard calculation.

It works like this: for an input of, say, 321.487… the messenger goes to both office 321 and 322 and collects their respective

outputs. Let's imagine that they are -14.3 and 12.5 respectively. All that's needed is a small calculation, which in this case will look like

$$-14.3 \times (1 - 0.487...) + 12.5 \times 0.487...$$

This is called ***linear interpolation*** and lets us construct continuous functions out of discrete data. There are other types of interpolation have have desirable properties, like "smoothness," which we'll learn about later.

It's very common in science to work with continuous domains by breaking them up into discrete pieces. As you'll see, an important strategy in calculus to to make such discrete pieces very close together, so that they resemble a continuum.

A table like REFERENCE TO THE TABLE PREVIOUS describes the general relationships between engine attributes. For instance, we might want to understand the relationship (if any) between RPM and engine mass, or relate the diameter (that is, "bore") and depth (that is, "stroke") of the cylinders to the power generated by the engine. Any single entry in the table doesn't tell us about such general relationships; we need to consider the rows and columns as a whole.

If you examined the relationship between engine power (`BHP`) and bore, stroke, and RPM, you will find that (as a rule) the larger the bore and stroke, the more powerful the engine. That's a ***qualitative*** description of the relationship. Most educated people are able to understand such a qualitative description. Even if they don't know exactly what "power" means, they have some rough conception of it.

Often, we're interested in having a ***quantitative*** description of a relationship such as the one (bore, stroke) $\rightarrow$ power. Remarkably, many otherwise well-educated people are uncomfortable with the idea of using quantitative descriptions of a relationship: what sort of language the description should be written with; how to perform the calculations to use the description; how to translate between data (such as in the table) and a quantitative description; how to translate the quantitative description to address a particular question or make a decision.

6.6 Exercises

Exercise XX.XX: evn4OP Handling data, plotting data

EXERCISE: Multiple graphics layers, more than 2? Perhaps data, contour plot,

EXERCISE: `Chirps` in `Zcalc`. Estimate the slope and intercept and plot that function as well. Or ask them to overplot the function temp = 40 + 0.9 * chirps.

6.7 Drill

Part II

Modeling

This is where I'll explain what the block is about and the overall goals.

7 Parameters

The pattern-book functions provide the modeler with a collection of shapes. They are not yet fully suited to represent real-world phenomena. To illustrate, consider Figure 7.1 which shows the number of officially confirmed COVID-19 cases in the US in March 2020.

The case count versus time in the COVID pandemic was widely and appropriately described as "exponential." So it seems appropriate alongside the data Figure Figure 7.1 shows the function cases$(t) \equiv e^t$ plotted as a magenta curve.

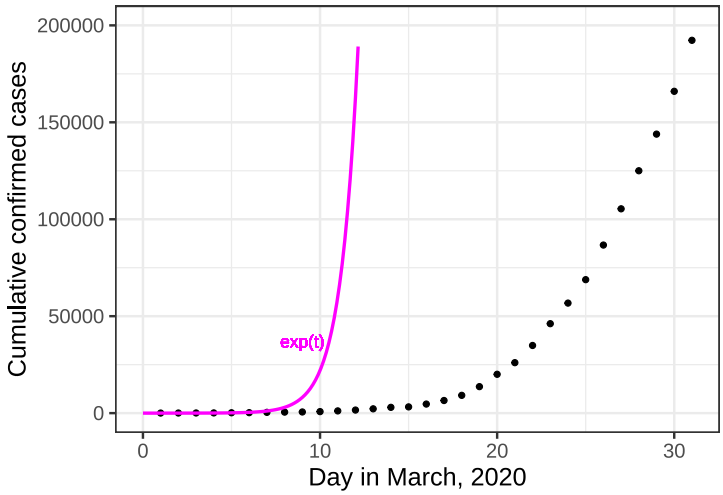


Figure 7.1: Cumulative officially confirmed COVID-19 cases during the month of March, 2020. The magenta curve is e^t.

There's an obvious mismatch between the data and the function e^t. Does this mean the COVID pattern is not exponential?

For the pattern book functions, the input is always a ***pure number***. For instance, we read $t = 10$ off the horizontal axis at which point $e^t\,|_{t=10} \approx 22000$, consistent with the curve shown in the graph.

But the place on the horizontal axis marked 10 does not correspond to the number 10. Rather, that place stands for "10

days," a quantity with units. Perhaps surprisingly, there is no such thing as $e^{10\,\text{days}}$. The reason for this will be detailed in Chapter (**chap-dimensions-and-units?**), but for now we'll simply point out that the domain of exp() is the set of real numbers, and real numbers don't have units.

If we want the input to cases(t) to be denominated in days, we'll have to convert t to a pure pure number (e.g. 10, not "10 days") *before* the quantity is handed off as the argument to exp(). We do this by introducing a ***parameter*** when we use the exponential function for describing relationships between quantities. The standard from for this is e^{kt} as opposed to e^t. The k parameter will be a quantity with units of "per-day." Suppose we set $k = 0.2$ per day. Then $k\,t\,|_{t=10 days} = 2$. This "2" is a pure number because the units on the 0.2 ("per day") and on the 10 (days) cancel out:

$$0.2\,\text{day}^{-1} \cdot 10\,\text{days} = 2\ .$$

The use of a parameter like k does more than handle the formality of converting input quantities into pure numbers. Having a choice for k allows us to stretch or compress the function to align with the data. Figure Figure 7.2 plots the modeling version of the exponential function to the COVID-case data:

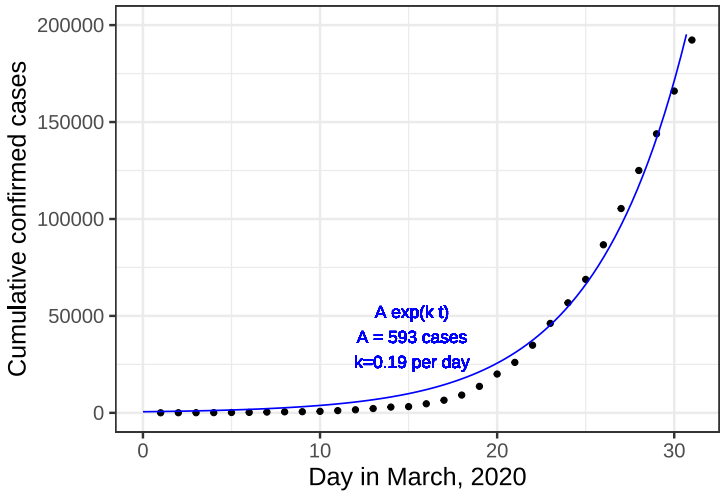
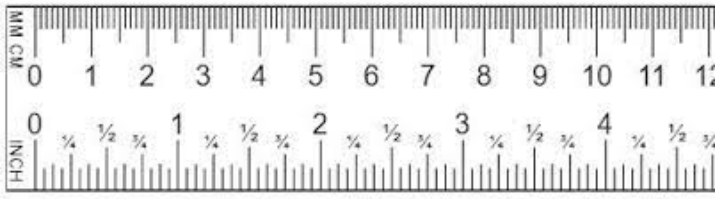


Figure 7.2: Using the function form Ae^{kt}, with parameters $k = 0.19$ per day and $A = 573$ cases, matches the COVID-case data well.

This chapter will introduce new functions, based on the pattern-book functions, but that have parameters so that the inputs and outputs can be **quantities** rather than pure numbers.

7.1 Parallel scales

At the heart of how we're going to use the pattern-book functions to model the relationship between quantities is the idea of conversion between one scale and another. Consider these everyday objects: a thermometer and a ruler.



Each object presents a read-out of what's being measured—temperature or length—on two different scales. At the same time, the objects provide a way to convert one scale to another.

A function gives the output for any given input. We represent the input value as a position on a number line—which we call an "axis"—and the output as a position on another output line, almost always drawn perpendicular to one another. But the two number lines can just as well be parallel to one another. To evaluate the function, find the input value on the input scale and read off the corresponding output.

We can translate the correspondance between one scale and the other into the form of a straight-line function. For instance, if we know the temperature in Fahrenheit (°F) and want to convert it to Celsius (°C) we have the following function:

$$C(F) \equiv \frac{5}{9}(F - 32) \ .$$

Similarly, converting inches to centimeters can be accomplished with

$$\text{cm(inches)} \equiv 2.54\,(\text{inches} - 0) \ .$$

Both of these scale conversion functions have the form of the straight-line function, which can be written as

$$f(x) \equiv ax + b \quad \text{or, equivalently as} \quad f(x) \equiv a(x - x_0) \ ,$$

where a, b, and x_0 are ***parameters***.

In Section Section 7.2, we'll use the $ax + b$ form of scale conversion, to scale the **input** to pattern-book functions, but we could equally well have used $a(x - x_0)$.

In Section Section 7.3 we'll introduce a second scale conversion function, for the **output** from pattern-book functions. That scaling will also be in the form of a straight-line function: $Ax + B$. The use of the lower-case parameter names (a, b) versus the upper-case parameter names (A, B) will help us distinguish the two different uses for scale conversion, namely ***input scaling*** versus ***output scaling***.

7.2 Input scaling

?@fig-tides-ri1 is based on the data frame `RI-tide` which is a minute-by-minute record of the tide level in Providence, Rhode Island (USA) for the period April 1 to 5, 2010. The `level` variable is measured in meters; the `hour` variable gives the time of the measurement in hours after midnight at the start of April 1.

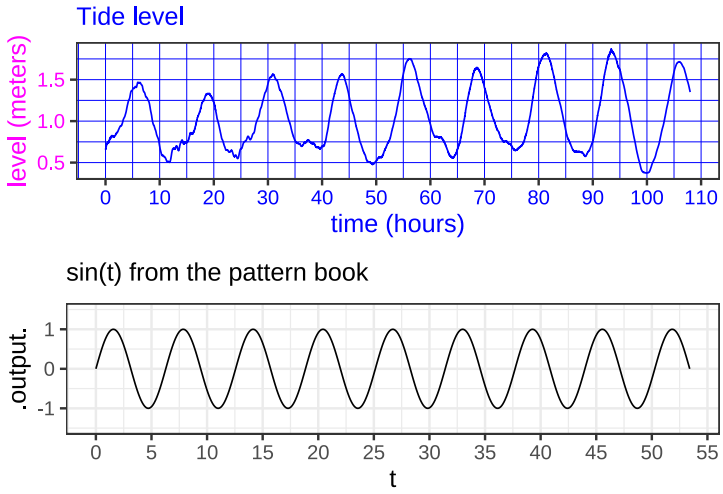


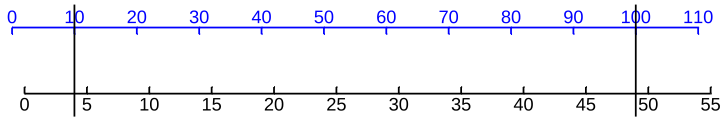
Figure 7.3: Tide levels oscillate up and down over time. This is analogous to the sin(t) pattern-book function.

The pattern-book sin() and the function level(*hour*) have similar shapes, so it seems reasonable to model the tide data as a sinusoid. However, the scale of the axes is different on the two graphs.

To model the tide with a sinusoid, we need to modify the sinusoid to change the scale of the input and output. First, let's look at how to accomplish the ***input scaling***. Specifically, we want the pure-number input t to the sinusoid be a function of the quantity *hour*. Our framework for this re-scaling is the straight-line function. We will replace the pattern-book input t with a function

$$t(hour) \equiv a\, hour + b\ .$$

The challenge is to find values for the parameters a and b that will transform the blue horizontal axis into the **black** horizontal axis, like this:



By comparing the two axes, we can estimate that $10 \rightarrow 4$ and $100 \rightarrow 49$. With these two coordinate points, we can find the straight-line function that turns blue into **black** by plotting the coordinate pairs $(0, 1)$ and $(100, 51)$ and finding the straight-line function that connects the points.

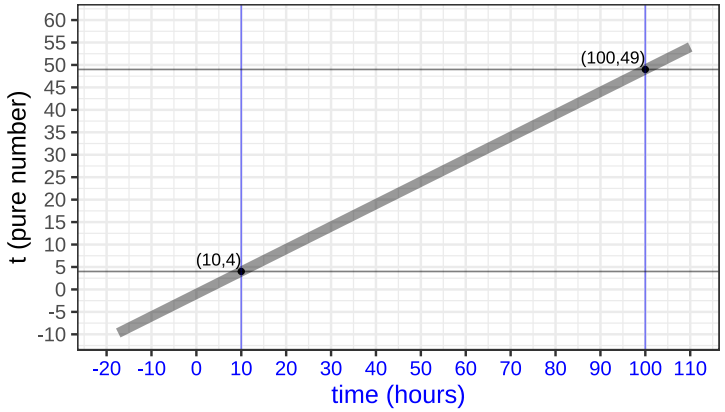


Figure 7.4: The input scaling function must transform 10 into 4 and transform 100 into 49 to properly arrange the time scale with the scale for the pattern-book function.

You can calculate for yourself that the function that relates blue to **black** is

$$t(time) = \underbrace{\frac{1}{2}}_{a} time \underbrace{-1}_{b}$$

Replacing the pure number t as the input to pattern-book $\sin(t)$ with the transformed $\frac{1}{2}time11$ we get a new function:

$$g(time) \equiv \sin\left(\frac{1}{2}time - 1\right) \ .$$

Figure **?@fig-tides-ri2** plots $g()$ along with the actual tide data.

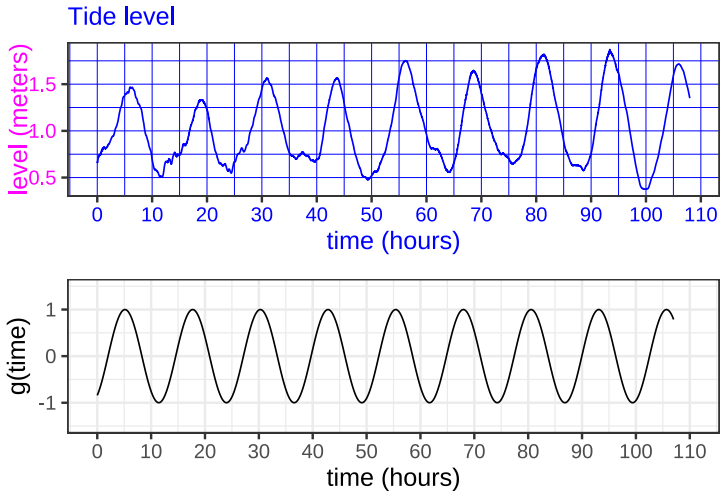


Figure 7.5: The sinusoid with input scaling (black) aligns nicely with the tide-level data.

7.3 Output scaling

Just as the natural input needs to be scaled before it reaches the pattern-book function, so the output from the pattern-book function needs to be scaled before it presents a result suited for interpreting in the real world.

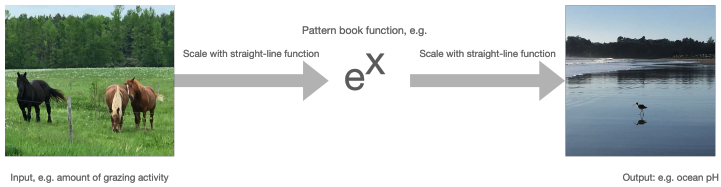


Figure 7.6: Natural **quantities** must be scaled to pure numbers before being suited to the pattern-book functions. The output from the pattern-book function is a pure number which is scaled to the natural **quantity** of interest.

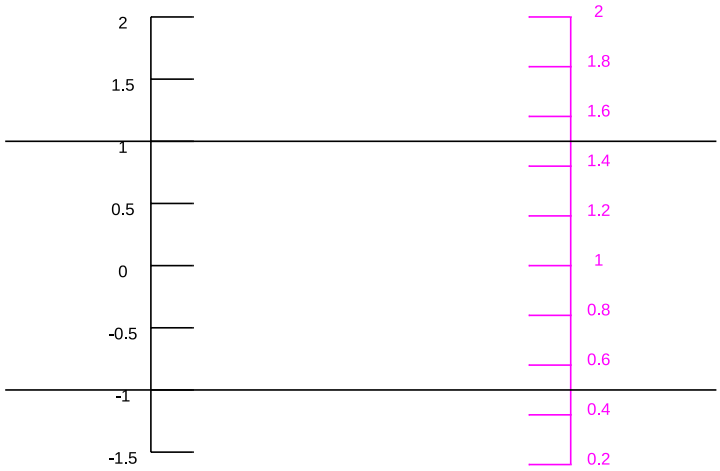
The overall result of input and output scaling is to tailor the pattern-book function so that it is ready to be used in the real world.

Let's return to Figure **?@fig-tides-ri2** which shows that the function $g(time)$, which scales the input to the pattern-book sinusoid, has a much better alignment to the tide data. Still, the vertical axes of the two graphs in the figure are not the same.

This is the job for ***output scaling***, which takes the output of $g(time)$ (bottom graph) and scales it to match the *level* axis on the top graph. That is, we seek to align the **black** vertical scale with the magenta vertical scale. To do this, we note that the range of the $g(time)$ is -1 to 1, whereas the range of the tide-level is about 0.5 to 1.5. The output scaling will take the straight-line form

$$\text{level}(time) = A\, g(time) + B$$

or, in graphical terms



We can figure out parameters A and B by finding the straight-line function that connects the coordinate pairs $(-1, 0.5)$ and $(1, 1.5)$ as in Figure Figure 7.7.

You can confirm for yourself that the function that does the job is

$$\text{level} = 0.5g(time) + 1\ .$$

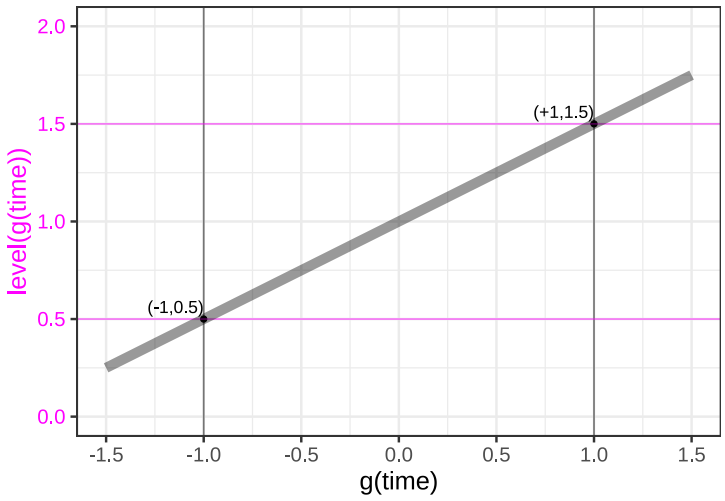


Figure 7.7: Finding the straight-line function that converts $-1 \rightarrow 0.5$ and converts $1 \rightarrow 1.5$

Putting everything together, that is, scaling both the input to pattern-book sin() and the output from pattern-book sin(), we get

$$\text{level}(time) = \underbrace{0.5}_{A} \sin\left(\underbrace{\frac{1}{2}}_{a} time \underbrace{-1}_{b}\right) + \underbrace{1}_{B}$$

7.4 A procedure for building models

We've been using pattern-book functions as the intermediaries between input scaling and output scaling, using this format.

$$f(x) \equiv Ae^{ax+b} + B \ .$$

We can use the other pattern-book functions—the gaussian, the sigmoid, the logarithm, the power-law functions—in exactly the same way. That is, the basic framework for modeling is this:

$$\text{model}(x) \equiv A\, g_{pattern\_book}(ax + b) + B \ ,$$

where $g_{pattern\_book}()$ is one of the pattern-book functions. To construct a basic model, you task has two parts:

1. Pick the specific pattern-book function whose shape resembles that of the relationship you are trying to model. For instance, we picked e^x for modeling COVID cases versus time (at the start of the pandemic). We picked $\sin(x)$ for modeling tide levels versus time.

2. Find numerical values for the parameters A, B, a, and b. In Chapter (**chap-fitting-by-eye?**) we'll show you some ways to make this part of the task easier.

It's remarkable that models of a very wide range of real-world relationships between pairs of quantities can be constructed by picking one of a handful of functions, then scaling the input and the output. As we move on to other Blocks in *MOSAIC Calculus*, you'll see how to generalize this to potentially complicated relationships among more than two quantities. That's a big part of the reason you're studying calculus.

7.5 Other formats for scaling

Often, modelers choose to use input scaling in the form $a(x-x_0)$ rather than $ax+b$. The two are completely equivalent when $x_0 = -b/a$. The choice between the two forms is largely a matter of convention. But almost always the output scaling is written in the format $Ay+B$.

For instance ...

For the COVID case-number data shown in Figure Figure 7.2, we found that a reasonable match to the data can be had by input- and output-scaling the exponential:

$$\text{cases}(t) \equiv \underbrace{573}_{A} e^{\overbrace{0.19}^{a} \; t} .$$

You might wonder why the parameters B and b aren't included in the model. One reason is that cases and the exponential function already have the same range: zero and upwards. So there's no need to shift the output with a parameter B.

Another reason has to do with the algebraic properties of the exponential function. Specifically,

$$e^{ax+b} = e^b e^{ax} = \mathcal{A}e^{ax}$$

where $\mathcal{A} \equiv e^b$.

In the case of exponentials, writing the input scaling in the form $e^{a(x-x_0)}$ can provide additional insight.

A bit of symbolic manipulation of the model can provide some additional insight. As you know, the properties of exponentials and logarithms are such that

$$Ae^{at} = e^{\log(A)}e^{at} = e^{at+\log(A)} = e^{a\left(t+\log(A)/a\right)} = e^{a(t-t_0)} \ ,$$

where

$$t_0 = -\log(A)/a = -\log(593)/0.19 = -33.6 \ .$$

You can interpret t_0 as the starting point of the pandemic. When $t = t_0$, the model output is $e^{k0} = 1$: the first case. According to the parameters we matched to the data for March, the pandemic's first case would have happened about 33 days before March 1, which is late January. We know from other sources of information, the outbreak began in late January. It's remarkable that even though the curve was constructed without any data from January or even February, the data from March, translated through the curve-fitting process, pointed to the start of the outbreak. This is a good indication that the exponential form for the model is fundamentally correct.

7.6 Parameterization idioms

English has many *idioms*, phrases to express an idea to those in the know, but will be confusing to others to take the phrase literally. Examples of such idioms (many of which come from Shakespeare's plays):

- break the ice

- it's Greek to me
- elbow room
- beat around the bush
- break a leg

Part of the process of learning a language is gaining familiarity with idioms.

The same is true for mathematics as it's spoken in different disciplines. Such idioms appear in the way input scaling is parameterized and the names used for the parameters. Often, the idiomatic parameterization is intended to make it easy to give a value to a parameter from easy-to-make observations. Knowing and using the idiom of mathematical notation will help you read and write mathematics more fluently.

Here are some input-scaling parameterizations that are used in practice.

Function	Written form	Parameter 1	Parameter 2
Exponential	e^{kt}	k "exponential constant"	Not used
Exponential	$e^{t/\tau}$	τ "time constant"	Not used
Exponential	$2^{t/\tau_2}$	τ_2 "doubling time"[1]	Not used
Power-law	$[x - x_0]^p$	x_0 x-intercept	exponent
Sinusoid	$\sin\left(\frac{2\pi}{P}(t - t_0)\right)$	P "period"	t_0 "time shift"
Sinusoid	$\sin(\omega t + \phi)$	ω "angular frequency"	ϕ "phase shift"
Sinusoid	$\sin(2\pi\omega t + \phi)$	ω "frequency"	ϕ "phase shift"
Gaussian	dnorm(x, mean, sd)	"mean" (center)	sd "standard deviation"
Sigmoid	pnorm(x, mean, sd)	"mean" (center)	sd "standard deviation"

[1] $-\tau_2$ is sometimes called the "half life."

Function	Written form	Parameter 1	Parameter 2
Straight-line	$mx + b$	m "slope"	b "y-intercept"
Straight-line	$m(x - x_0)$	m "slope"	x_0 "center"

7.7 Exercises

Exercise 7.2: uKCIE unassigned

Exercise 7.3: BLECL unassigned

Exercise 7.1: MWLCS unassigned

Exercise 7.5: FKLEU unassigned

Exercise XX.XX: amCjZG input & output scaling

Exercise XX.XX: 2iItLg Input and output scaling

8 Assembling functions

When we need a new function for some purpose, we practically always build it out of existing functions. For instance, a parameterized function like

$$f(x) \equiv A \sin\left(\frac{2\pi}{P}x\right) + B$$

is built by assempling together a straight-line input scaling, a pattern-book sin() function, and another straight-line function for scaling the output from sin(). This is an example of ***function composition*** where functions are "layered," the output of one function being given as the input to another. For instance, $f(x)$ is put together from three composed functions, output(), sin(), input() where

$$f(x) = \text{output}(sin(\text{input}(x))$$

where

$$\text{output}(x) \equiv Ax + B \quad \text{and} \quad \text{input}(x) \equiv ax + b$$

In this chapter, we'll review three general frameworks for combining functions: linear combination, composition, and multiplication. You have almost certainly seen all three of these frameworks in your previous mathematical studies, although you might not have known that they have names.

8.1 Linear combination

One of the most widely used sorts of combination is called a ***linear combination***. The mathematics of linear combination is, it happens, at the core of the use of math in applications, whether that be constructing a Google-like search engine or

analyzing medical data to see if a treatment has a positive effect.

You've worked for many years with one kind of linear combination: polynomials. No doubt you've seen functions[1] like

$$f(x) \equiv 3x^2 + 5x - 2$$

There are three pattern-book functions in this polynomial. In polynomials the functions being combined are all power-law functions: $g_0(x) \equiv 1$, $g_1(x) \equiv x$, and $g_2(x) \equiv x^2$. With these functions defined, we can write the polynomial $f(x)$ as

$$f(x) \equiv 3g_2(x) + 5g_1(x) - 2g_0(x)$$

Each of the functions is being scaled by a quantity—3, 5, and -2 in this example—and the scaled functions are added up. That's a linear combination; scale and add. (Later, we'll see that the ***scalars*** generally come with units. So we might well have a metric polynomial and an equivalent traditional-unit polynomial. Just wait.)

There are other places where you have seen linear combinations:

- The parameterized ***sinusoid***

  $$A \sin\left(\frac{2\pi}{P}t\right) + B$$

  is a linear combination of the functions $h_1(t) \equiv \sin\left(\frac{2\pi}{P}t\right)$ and $h_2(t) \equiv 1$. The linear combination is $A\,h_1(t) + B\,h_2(t)$.
- The parameterized ***exponential***

  $$Ae^{kt} + B$$

  The functions being combined are e^{kt} and 1. The scalars are, again, A and C.

[1] It's likely that you saw polynomials as things to be factored, rather than as functions taking an input and producing an output. So they were written as *equations*: $3x^2 + 5x - 2 = 0$.

- The straight line function, such as output$(x) \equiv Ax + B$ and input$(x) \equiv ax + b$. The functions being combined are x and 1, the scalars are a and b.

Note that neither the parameterized exponential or the parameterized sinusoid is a polynomial simply because it is not constructed exclusively from monomials.

There are a few reasons for us to be introducing linear combinations here.

1. You will see linear combinations everywhere once you know to look for them.
2. There is a highly refined mathematical theory of linear combinations that gives us powerful ways to think about them as well as computer software that can quickly find the best scalars to use to match input-output data.
3. The concept of linear combination generalizes the simple idea that we have been calling "scaling the output." From now on, we'll use the linear-combination terminology and avoid the narrower idea of "scaling the output."
4. Many physical systems are described by linear combinations. For instance, the motion of a vibrating molecule or a helicopter in flight or a building shaken by an earthquake are described in terms of simple "modes" which are linearly combined to make up the entire motion. More down to Earth, the timbre of a musical instrument is set by the scalars in a linear combination of pure tones.
5. Many modeling tasks can be put into the framework of choosing an appropriate set of simple functions to combine and then figuring out the best scalars to use in the combination. (Generally, the computer does the figuring.)

8.2 Function composition

To ***compose*** two functions, $f(x)$ and $g(x)$, means to apply one of the functions to the output of the other. "$f()$ composed with $g()$" means $f(g(x))$. This is generally very different from "$g()$ composed with $f()$" which means $g(f(x))$.

For instance, suppose you have recorded the outdoor temperature over the course of a day and packaged this into a function AirTemp(t): temperature as a function of time t. Your digital thermometer uses degrees Celsius, but you want the output units to be degrees Kelvin. The conversion function is

$$\text{CtoK}(C) \equiv C + 273.15$$

Notice that CtoK() takes temperature in $^\circ C$ as input. With this, we can write the "Kelvin as a function of time" as

$$\text{CtoK}\left(\text{AirTemp}(t)\right)$$

It's important to distinguish the above time $\rightarrow$ Kelvin function from something that looks very much the same but is utterly different: $\text{AirTemp}\left(\text{CtoK}(C)\right)$. In the first, the input is time. In the second, it is temperature in celsius.

Here is a model of the length of daylight (in hours) as a function of latitude L and the declination angle δ of the sun.

$$\text{daylight}(L, \delta) \equiv \frac{2}{15} \arccos\left(-\tan(L) * \tan(\delta)\right)$$

The declination angle is the latitude of the point on the earth's surface pierced by an imagined line connecting the centers of the earth and the sun. On the summer solstice, the longest day of the year, it is 23.44°. On day, where midnight before January 1 is $day = 0$ and the end of December 31 is $day = 365.25$, the declination is

$$\delta(day) = 23.44 \sin\left(\frac{2\pi}{365.25}(day - 9)\right) \ ,$$

a composition of sin() with the straight-line function $\frac{2\pi}{365.25}(day - 9)$.

Composing day_length(L, δ) onto $\delta(day)$ gives the length of daylight as a function of day of the year:

$$\text{light}(L, day) \equiv \frac{2}{15} \arccos\left(-\tan(L) * \tan(\delta(day))\right) \ .$$

Function composition enables us to transform a function that takes one kind of thing as input (say, declination) and turn it

into a function that takes another kind of thing as input (say, day of the year).

Math in the world ...

Income inequality is a matter of perennial political debate. In the US, most people support Social Security, which is an income re-distribution programming dating back almost a century. But other re-distribution policies are controversial. Some believe they are essential to a healthy society, others that the "cure" is worse than the "disease."

Whatever one's views, it's helpful to have a way to quantify inequality. There are many ways that this might be done. A mathematically sophisticated one is called the ***Gini coefficient***.

Imagine that society was divided statistically into income groups, from poorest to richest. Each of these income groups consists of a fraction of the population and has, in aggregate, a fraction of the national income. Poor people tend to be many in number but to have a very small fraction of income. Wealthy people are few in number, but have a large fraction of income. The table shows data for US households in 2009:[2]

group label	population	aggregate income	cumulative income
poorest	20%	3.4%	3.4%
low-middle	20%	8.6%	12.0%
middle	20%	14.6%	26.6%
high-middle	20%	23.2%	47.8%
richest	20%	50.2%	100.0%

The ***cumulative*** income shows the fraction of income of all the people in that group or poorer. The cumulative population adds up the population fraction in that row and previous rows. So, a cumulative population of 60% means "the poorest 60%

[2]These data, as well as the general idea for the topic come from La Haye and Zizler (2021), "The Lorenz Curve in the Classroom", *The American Statistician*, 75(2):217-225

of the population" which, as the table shows, earn as a group 14.6% of the total income for the whole population.

A function that relates the cumulative population to the cumulative income is called a ***Lorenz function***. The data are graphed in Figure 8.1 and available as the `US_income` data frame in the SANDBOX. Later, in Figure 8.2, we'll fit parameterized functions to the data.

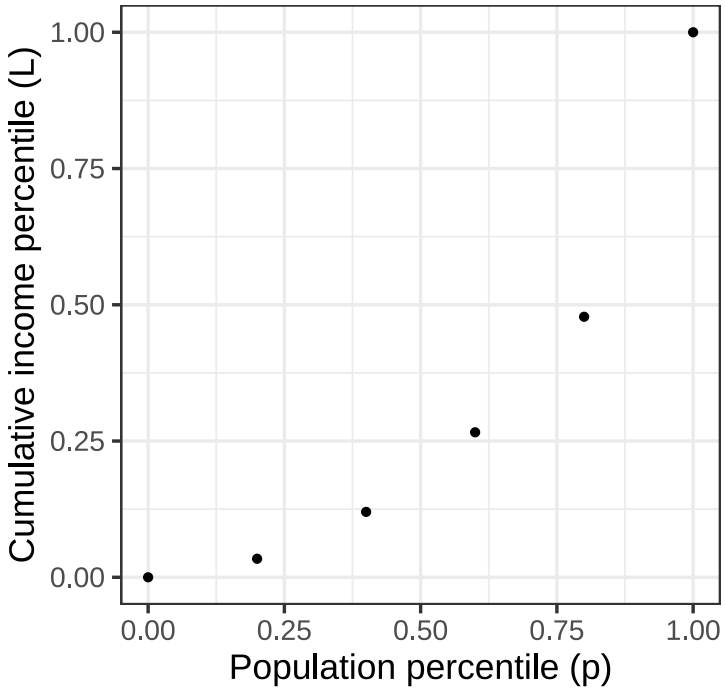


Figure 8.1: Data on household incomes in the US in 2009.

Lorenz curves must:

- Be concave up, which amounts to saying that the curve gets steeper and steeper as the population percentile increases. (Why? Because at any point, poorer people are to the left and richer to the right.)
- Connect (0,0) to (1, 1).

Calling the income percentile L a function of the population percentile p, a Lorenz function is $L(p)$ that satisfies the requirements in the previous paragraph. Here are some functions that meet the requirements:

- $L_b(p) \equiv p^b$ where $1 \leq b$.

- $L_q(p) \equiv 1 - (1-p)^q$ where $0 < q \leq 1$

Notice that each of these functions has just one parameter. It seems implausible that the workings of a complex society can be summarized with just one number. We can use the curve-polishing techniques that will be introduced in (**chap-fitting-polishing?**) to find the "best" parameter value to match the data.

```
Lb <- fitModel(income ~ pop^b, data = Income, start=list(b=1.5))
Lq <- fitModel(income ~ 1 - (1-pop)^q, data = Income, start=list(q=0.5))
```

Figure 8.2 compares the fitted functions to the data.

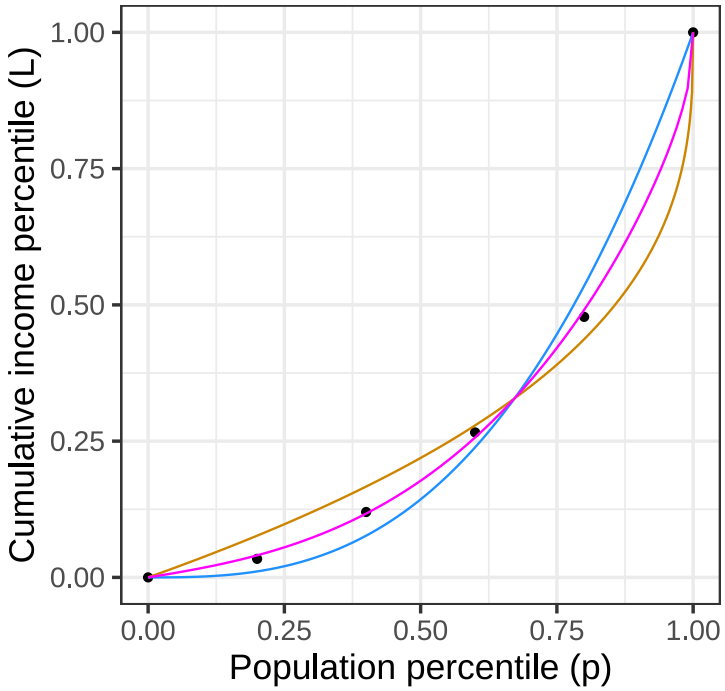


Figure 8.2: Lorenz curves $L_b(p)$ (blue) and $L_q(p)$ (magenta) fitted to the household income data.

Neither form $L_b(p)$ or $L_q(p)$ gives a compelling description of the data. Where should we go from here?

We can provide more parameters by constructing more complicated Lorenz functions. Here are two ways to build a new Lorenz function out of an existing one:

- The product of any two Lorenz functions, $L_1(p)L_2(p)$ is itself a Lorenz function.

- A linear combination of any two Lorenz functions, $aL_1(p) + (1-a)L_2(p)$, so long as the scalars add up to 1, is itself a Lorenz function. For instance, the magenta curve in Figure 8.2 is the linear combination of 0.45 times the tan curve plus 0.55 times the blue curve.

Question: Is the composition of two Lorenz functions a Lorenz function? That is, does the composition meet the two requirements for being a Lorenz function?

To get started, figure out whether or not $L_1(L_2(0)) = 0$ and $L_1(L_2(1)) = 1$. If the answer is yes, then we need to find a way to compute the concavity of a Lorenz function to determine if the composition will always be concave up. We'll need additional tools for this. We'll introduce these in Block 2.

8.3 Function multiplication

The third in our repertoire of methods for making new function out of old is plain old multiplication. With two functions $f(x)$ and $g(x)$, the product is simply $f(x)g(x)$.

It's essential to distinguish between function multiplication and function composition:

$$\underbrace{f(x)g(x)}_{\text{multiplication}} \qquad \underbrace{f(g(x)) \text{ or } g(f(x))}_{\text{composition}}$$

In function composition, only one of the functions—the ***interior function*** is applied to the overall input, x in the above example. The ***exterior function*** is fed its input from the output of the interior function.

In multiplication, each of the functions is applied to the input individually. Then their outputs are multiplied to produce the overall output.

In function composition, the order of the functions matters: $f(g(x))$ and $g(f(x))$ are in general completely different functions.

In function multiplication, the order doesn't matter because multiplication is ***commutative***, that is, if $f()$ and $g()$ are the functions to be multiplied $f(x) \times g(x) = g(x) \times f(x)$.

FOR INSTANCE ...

Transient vibration

A guitar string is plucked to produce a note. The sound is, of course, vibrations of the air created by vibrations of the string.

After plucking, the note fades away. An important model of this is a sinusoid (of the correct period to correspond to the frequency of the note) times an exponential.

Function multiplication is used so often in modeling that you'll see it in many modeling situations. Here's one example that is important in physics and communication: the ***wave packet***. Overall, the wave packet is a localized oscillation as in Figure 8.3.

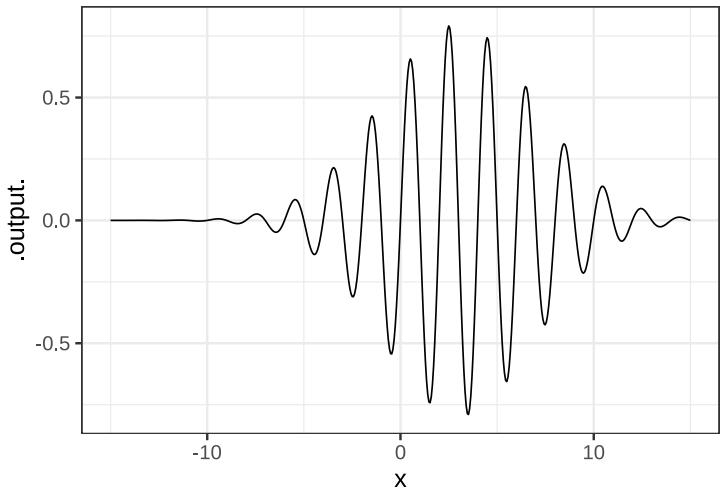


Figure 8.3: A *wave packet* constructed by multiplying a sinusoid and a gaussian function.

This is the product of two simple functions: a gaussian times a sinusoid.

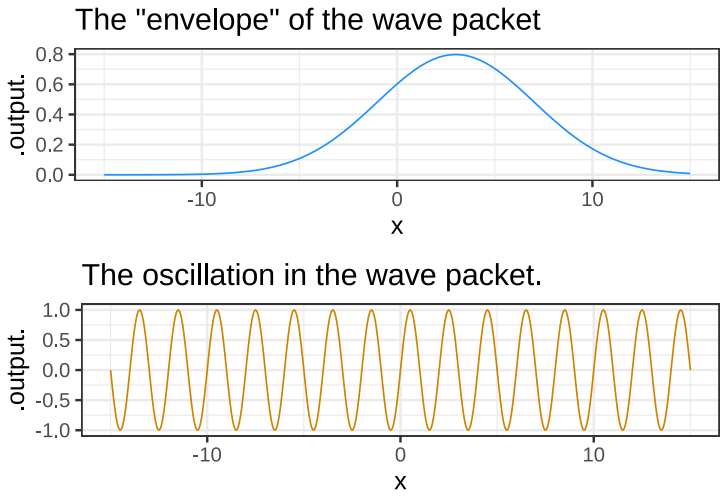


Figure 8.4: The two components of the wave packet in Figure 8.3

MATH IN THE WORLD ...

The initial rise in popularity of the social media platform Yik Yak was exponential. Then popularity leveled off, promising a steady, if static, business into the future. But, the internet being what it is, popularity collapsed to near zero and the company closed.

On way to model this pattern is by multiplying a sigmoid by an exponential.(See Figure 8.5.)

Functions constructed as a ***product*** of simple functions can look like this in tradition notation:

$$h(t) \equiv \sin(t)e^{-t}$$

and like this in computer notation:

```
h <- makeFun(sin(t)*exp(-t) ~ t)
```

8.4 Watch your domain!

Each of our pattern-book functions, with two exceptions, has a domain that is the entire number line $-\infty < x < \infty$. No

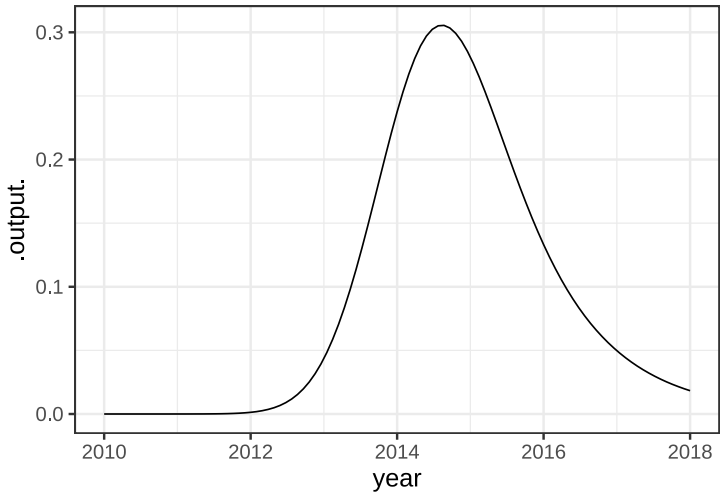


Figure 8.5: Subscriptions to the web messaging service Yik Yak grew exponentially in 2013 and 2014, then collapsed. The company closed in 2017.

matter how big or small is the value of the input, the function has an output. Such functions are particularly nice to work with, since we never have to worry about the input going out of bounds.

The two exceptions are:

1. the logarithm function, which is defined only for $0 < x$.
2. some of the power-law functions: x^p.

    - When p is negative, the output of the function is undefined when $x = 0$. You can see why with a simple example: $g(x) \equiv x^{-2}$. Most students had it drilled into them that "division by zero is illegal," and $g(0) = \frac{1}{0}\frac{1}{0}$, a double law breaker.
    - When p is not an integer, that is $p \neq 1, 2, 3, \cdots$ the domain of the power-law function does not include negative inputs. To see why, consider the function $h(x) \equiv x^{1/3}$.

R/MOSAIC COMMANDS

It can be tedious to make sure that you are on the right side of the law when dealing with functions whose domain is not the whole number line. The designers of the hardware that does computer arithmetic, after several decades of work, found a clever system to make it easier. It's a standard part of such hardware that whenever a function is handed an input that is not part of that function's domain, one of two special "numbers" is returned. To illustrate:

```
sqrt(-3)
## [1] NaN
(-2)^0.9999
## [1] NaN
1/0
## [1] Inf
```

`NaN` stands for "not a number." Just about any calculation involving `NaN` will generate `NaN` as a result, even those involving multiplication by zero or cancellation by subtraction or division.[3] For instance:

```
0 * NaN
## [1] NaN
NaN - NaN
## [1] NaN
NaN / NaN
## [1] NaN
```

Division by zero produces `Inf`, whose name is reminiscent of "infinity." `Inf` infiltrates any calculation in which it takes part:

```
3 * Inf
## [1] Inf
sqrt(Inf)
## [1] Inf
0 * Inf
## [1] NaN
Inf + Inf
## [1] Inf
```

[3] One that does produce a number is `NaN^0`.

```
Inf - Inf
## [1] NaN
1/Inf
## [1] 0
```

To see the benefits of the `NaN` / `Inf` system let's plot out the logarithm function over the graphics domain $-5 \leq x \leq 5$. Of course, part of that graphics domain, $-5 \leq x \leq 0$ is not in the domain of the logarithm function and the computer is entitled to give us a slap on the wrists. The `NaN` provides some room for politeness.

Open an R console and see what happens when you make the plot.

```
slice_plot(log(x) ~ x, domain(x=c(-5,5)))
```

8.5 Splitting the domain

Let's consider a familiar mathematical function: the absolute-value function:

$$abs(x) = |x|$$

Written this way, the definition of $abs()$ is a tautology. Unless you already know what $|x|$ means, you will have no clue what's going on.

So, instead, let's define $abs()$ in terms of pattern-book functions and scaling. It will look like this: But with the ability to divide the domain into pieces, we gain access to a less mysterious sort of arithmetic operation and can re-write

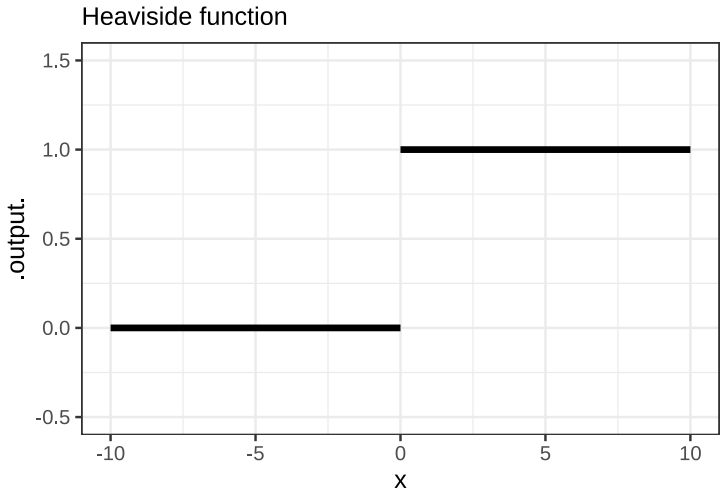
$$abs(x) \equiv \begin{cases} x & \text{for } 0 \leq x \\ -x & \text{otherwise} \end{cases}$$

This is an example of a ***piecewise function***, that is a function whose domain is split into two or more intervals and defined by

different formulas on those intervals. In the conventional mathematical notation, there is a large { followed by two or more lines. Each line gives a formula for that part of the function and indicates to which interval the formula applies.

Another piecewise function widely used in technical work, but not as familiar as $abs()$ is the ***Heaviside function***: Less familiar is the ***Heaviside function*** which has important uses in physics and engineering:

$$\text{Heaviside}(x) \equiv \begin{cases} 1 & \text{for } 0 \leq x \\ 0 & \text{otherwise} \end{cases}$$



The traditional piecewise notation involving { is not directly useful as computer notation. In R, a handy way to define a piecewise function uses the R function `ifelse()` whose name is remarkably descriptive. The `ifelse()` function takes three arguments. The first is the question to be asked, the second is the value to return if the answer is "yes," and the third is the value to return for a "no" answer. Here's an example:

```
H <- makeFun(ifelse(0 <= x, 1, 0) ~ x)
```

What takes getting used to here is the expression `0 <= x`. That expression is a ***question***; it is not a statement of fact.

The table shows computer notation for some common sorts of questions.

R notation	English
`x > 2`	"Is x greater than 2?"
`y >= 3`	"Is y greater than or equal to 3?"
`x == 4`	"Is x exactly 4?"
`2 < x & x < 5`	"Is x between 2 and 5?"[4]
`x < 2 | x > 6`	"Is x either less than 2 or greater than 6?"
`abs(x-5) < 2`	"Is x within two units of 5?"

The vertical gap between the two pieces of the Heaviside function is called a ***discontinuity***. Intuitively, you cannot draw a discontinuous function ***without lifting the pencil from the paper***. The Heaviside function has a discontinuity at $x = 0$.

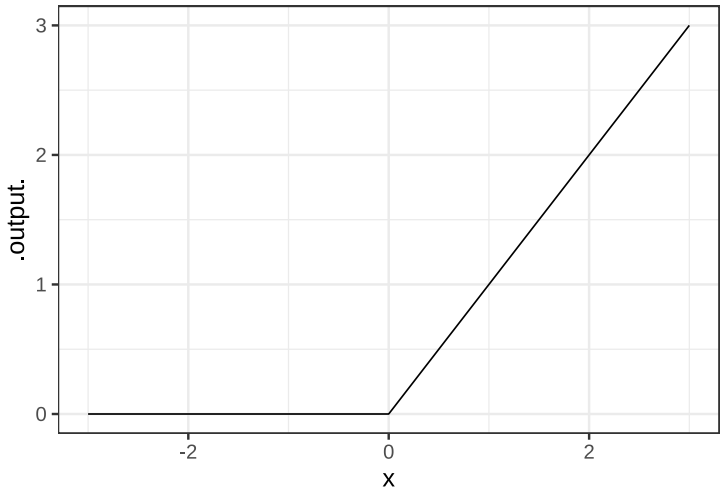
Similarly, the ***ramp function*** is a kind of one-sided absolute value:

$$\text{ramp}(x) \equiv \begin{cases} x & \text{for } 0 \leq x \\ 0 & \text{otherwise} \end{cases}$$

Or, in computer notation: ::: {.cell .column-margin layout-align="center" fig.showtext='true'}

```
ramp <- makeFun(ifelse(0 < x, x, 0) ~ x)
slice_plot(ramp(x) ~ x, domain(x=c(-3, 3)))
```

[4] Literally, "Is x both greater than 2 and less than 5?"



A linear combination of two input-shifted ramp functions gives a piecewise version of the sigmoid. ::: {.cell .column-margin layout-align="center" fig.showtext='true'}

```
sig <- makeFun(ramp(x+0.5) - ramp(x-0.5) ~ x)
slice_plot(sig(x) ~ x, domain(x=c(-3, 3)), npts=501)
```

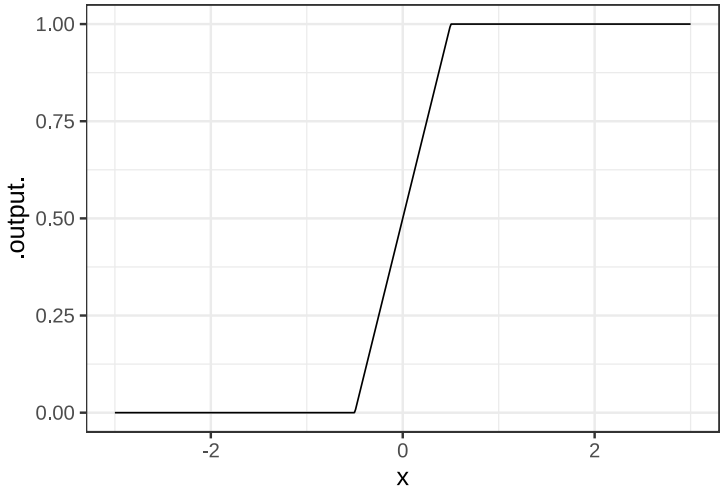


Figure 8.6 is a graph of monthly natural gas use in the author's household versus average temperature during the month. (Natural gas is measured in cubic feet, appreviated *ccf*.)

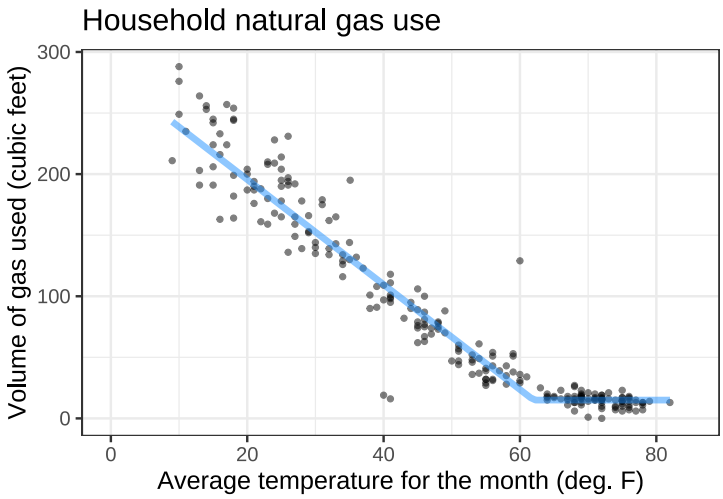


Figure 8.6: The amount of natural gas used for heating the author's home varies with the outside temperature.

The graph looks somewhat like a hockey stick. A sloping straight-line dependence of ccf on temperature for temperatures below 60°F and constant for higher temperatures. The shape originates from the dual uses of natural gas. Gas is used for cooking and domestic hot water, the demand for which is more of less independent of outdoor temperature at about 15 ccf per month. Gas is also used for heating the house, but that's needed only when the temperature is less than about 60°F.

We can accomplish the hockey-stick shape with a linear combination of the ramp() function and a constant. The ramp function represents gas used for heating, the constant is the other uses of gas (which are modeled as not depending on temperature. Overall, the model is

$$\text{gas}(x) \equiv 4.3\,\text{ramp}(62 - x) + 15 \ .$$

Even simpler is the model for the other uses of natural gas:

$$\text{other}(x) \equiv 15 \ .$$

8.6 Exercises

Exercise XX.XX: FCXT1L function composition

Exercise XX.XX: 3N3hCR function composition, function multiplication, linear combination

Exercise XX.XX: HMTG1w function multiplication

Use `datasets::co2` as an example of a product of functions. Maybe pull out a smoother as the baseline and see if the amplitude changes with time. Or maybe look at successive differences, fit a sine with a time dependent amplitude.

Exercise 13.05: EDKYV unassigned

Exercise 13.07: E9e7c6 unassigned

Exercise XX.XX: mtHyvJ asymptotes

9 Fitting and polishing

In Chapter @ref(assembling) we discussed a framework for building on pattern-book functions by use of ***input scaling*** and ***output scaling***, as with:

$$\text{tide}(t) \equiv A \sin\left(a(t - t_0)\right) + B \ .$$

In this chapter, we'll build on both these ideas to make them easier to use:

- input scaling will become ***parameter selection***: We'll give standard procedures that allow you to determine by eye the input-scaling parameters for the functions exp(), sin(), gaussian(), and sigmoid(). These will be similar in spirit to the procedures you learned in high-school for finding the parameters of the straight-line function from visually identifiable aspects such as the slope and y-intercept.
- output scaling will become finding ***linear combinations***: We'll generalize the idea beyond the A/B format used in Chapter @ref(assembling) to include finding linear combinations of any numbers of functions, e.g. $Af(x)+Bg(x)+Ch(y)+....$ The importance of building linear combinations can hardly be overstated. In fact, Block 5 of *MOSAIC Calculus* is entirely about this.

Taken together, the tools of parameter selection and finding linear combination support a standard process for building model functions.

9.1 Parameter names

The functions we will work with in this chapter either have no parameters (e.g. one() or identity()) of have standard formats

and names for parameters.

The gaussian and sigmoid share the same parameters: "mean" and "sd" (short for "standard deviation").

The exponential function is written in several different forms with different parameter names. Perhaps the most standard is $\exp(kt)$ with "k" as the parameter name. Closely related is the form $\exp(-kt)$, which is used especially for exponential decay. Another form often seen is $\exp(-t/\tau)$, where the parameter τ (a Greek letter pronounced "tau") is the "time constant." In practice, as you'll see, it's common to estimate quantities called the "doubling time" or "half-life," which are relatively easy to read from a graph, but then have to be converted the k or τ form.

For the sinusoid, we will often use the form $\sin\left(\frac{2\pi}{P}(t-t_0)\right)$, where the parameter P is the "period" of oscillation. But in fields such as engineering or physics one frequently encounters the sinusoid parameterized by "frequency" or "angular frequency" or "wave number" or "wavelength." All of these get at the same characteristic as "period," but in ways that are more or less convenient depending on the application.

9.2 Straight-line function

9.3 Gaussian and Sigmoid

9.4 Sinusoid

9.5 Exponential

9.6 FROM EARLIER INTRO

introduced three important techniques for constructing new functions out of existing ones:

- Composition, e.g. $f(g(x))$

- Multiplication, e.g. $f(x)g(x)$ or, a function with two inputs, $f(y)g(x)$
- Linear combination, e.g. $Af(x) + B\,g(x)$ or, a function with two inputs, $Af(y) + B\,g(x)$

Don't be misled by the short examples: any of the three techniques can be used to combine any number of combinations. In Chapter @ref(low-order), for instance, you'll see instances when we flexibly build new functions with two inputs by linearly combining six standard functions.

This chapter is about the uses of linear combination

We've discussed ***shifting the baseline*** of the exponential and sinusoid functions by adding a constant to the basic modeling function, like this:

$$f(t) \equiv Ae^{kt} + Cg(t) \equiv A\sin\left(\frac{2\pi}{P}t\right) + C$$

We've called this output scaling. It's an example of a much more general and powerful way of contructing modeling functions called ***linear combinations***.

In a linear combination, you start with one or more basic functions. For discussion, let's call these $f_1(t)$, $f_2(t)$, $f_3(t)$ and so on. In making a linear combination, multiply each of the basic functions by some quantity and add the results together:

$$\text{linear combination of } f_1(t) \text{ and } f_2(t): \quad A_1 f_1(t) + A_2 f_2(t)$$

The quantities A_1 and A_2 are called ***scalars*** because they **scale** the functions. (In Chapter @ref(vectors) you will see that we make a distinction between a **scalar**, which is a single quantity, and a **vector**, which is a set of scalars.)

It's common to take linear combinations of functions with different inputs, for example

$$h(x, y, t) \equiv 4e^{-kt} + 7y + 2\,\text{pnorm}(x - 3) - 19$$

In fields like statistics and economics and other social sciences, as well as clinical medical research, a great number of modeling

techniques involve such combinations of functions of various inputs.

In engineering and physics, an important class of functions involves a linear combination of many sinusoids of different periods, e.g.

$$\text{signal}(t) \equiv A_1 \sin\left(\frac{2\pi}{P_1}t\right) + A_2 \sin\left(\frac{2\pi}{P_2}t\right) + A_3 \sin\left(\frac{2\pi}{P_3}t\right) + \cdots$$

Linear combination of functions provides a powerful and flexible general-purpose modeling technique in part because many physical systems seem to work this way and in part because the methods for finding the scalars—A_1, A_2, and so on—have an extremely strong theory and fast computer implementations that automatically solve the problem once the modeler has selected the functions she wants to combine. In Block 4, we use the name ***target problem*** to refer the problem of finding scalars to match as well as possible a linear combination to data.

An expression like $Ae^{kx} + C$ combines the exponential (obviously!) and a second function that isn't immediately obvious: the constant function $\text{constant}(x) \equiv 1$. It might be overkill to re-write the expression as $Ae^{kx} + C\,\text{constant}(x)$ but get in the habit of seeing the constant function as a constant presence in linear combinations!

Polynomials are an important example of linear combinations. Something like $p(x) \equiv a + bt + ct^2$ is a linear combination of the constant function, the proportional function, and the square function.

9.7 Functions with multiple inputs

We can use ***linear combination*** and ***function multiplication*** to build up custom functions from the basic modeling functions. Similarly, linear combination and function multiplication provide ways to construct functions of multiple inputs.

9.8 f(x) times g(t)

For example, when a guitar string is at rest it forms a straight line connecting its two fixed ends: one set by finger pressure along the neck of the guitar and the other at the bridge near the center of the guitar body. When a guitar string is plucked, its oscillations follow a sinusoid pattern of **displacement**. With the right camera and lighting setup, we can see these oscillations in action:

For a string of length L, the string displacement is a function of position x along the string and is a linear combination of functions of the form

$$g_k(x) \equiv \sin(k\pi x/L)$$

where k is an integer. A few of these functions are graphed in Figure @ref(fig:guitar-string-modes) with $k = 1$, $k = 2$, and $k = 3$.

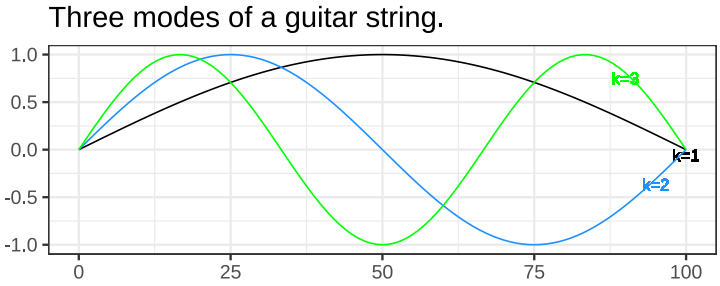


Figure 9.1: Vibrational modes of a guitar string.

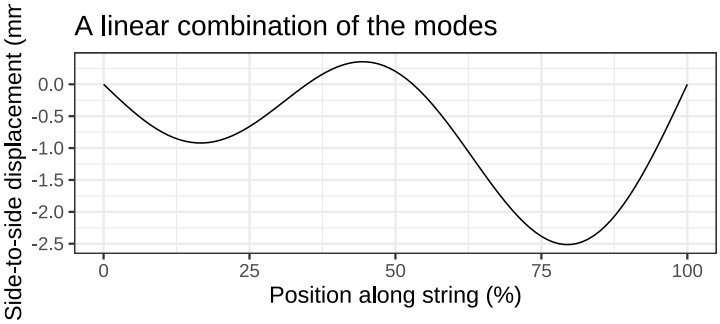


Figure 9.2: Vibrational modes of a guitar string.

Shapes of the sort in Figure @ref(fig:guitar-string-modes) are a stop-motion flash snapshot of the string. The string's shape also changes in time, so the string's displacement is a function of both x and t. The displacement itself is a sinusoid whose time period depends on the length and tension of the string as well as the number of cycles of the spatial sine:

$$g_k(x,t) \equiv \sin(\frac{k\pi}{L}x)\ \ \sin(\frac{k\pi}{P}t)$$

Figure @ref(fig:string-motion) shows a few snapshots of the 1.5 cycle string at different moments in time, and the motion of the linear combination.

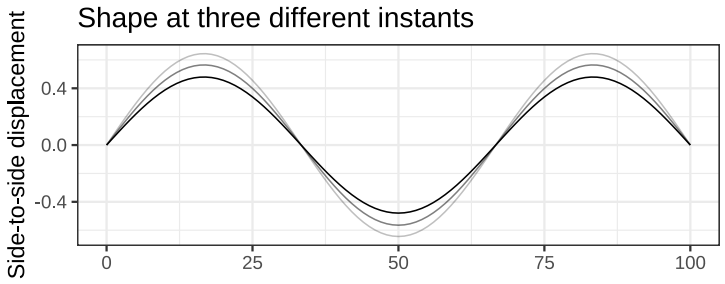


Figure 9.3: String position changes over time.

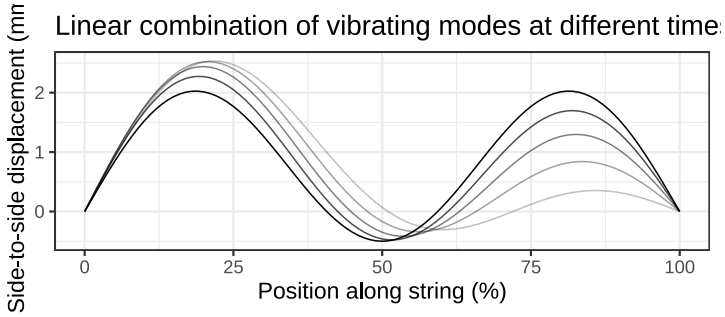


Figure 9.4: String position changes over time.

Why did you?

We left function ***composition*** out of the list of ways to build multivariable functions out of simpler functions with a single input.

For instance, consider the two functions $f(x)$ and $g(t)$. The composition $f(g(t))$ has only **one** input: t. Similarly, $g(f(x))$ has only one input: x.

9.9 Exercises

Exercise 12.1: ICLSE unassigned

Exercise 12.2: EDKKW unassigned

Exercise 12.4: dB1r5F unassigned

Exercise 12.5: drawing unassigned

Exercise 12.6: daylength unassigned

Exercise 12.7: kelxlB unassigned

Exercise 12.8: ticlw unassigned

10 Rate of change

For our purposes, the ***definition of calculus*** is

> *The use of functions to model and explore continuous change*

The agenda of this chapter is to give specific mathematical meaning to the word "change."

10.1 Change and slope

You have an solid, intuitive sense of what "change" means. In mathematics, and especially the mathematics of functions, change has a very simple meaning that you have already touched on in your previous math education.

The word that encapsulates "change" in high-school math is ***slope***. For instance, you've undoubtedly had to calculate the slope of a straight line in a graph. You learned about "rise" and "run" and how to read them from a graph or from a formula. The slope is the ratio: rise over run.

Everyone has a intuitive sense of the slope of a road or of a hillside. You learned to apply this intuition to reading graphs and the slope of a line. We'll exploit the intuitive ability to read a landscape in order to introduce abstract mathematical ideas in a down-to-earth setting.

Mathematical modelers learn to think of "slope" abstractly, not just as referring to the incline of a road. For instance, the population of a country can change, as can the number of new cases of an epidemic disease, the temperature of a cup of coffee, or the distance from Earth of a spacecraft. A major part of learning calculus is generalizing and abstracting the mathematical concept of which "slope" is an example.

10.2 Continuous change

Most people are comfortable with the ideas of daily changes in temperature or monthly changes in credit-card debt or quarterly changes in the unemployment rate or annual changes in the height of a child. Such things are easy to record in, say, a spreadsheet. For example, as this paragraph is being written, the weather forecast for the next several days (in southeastern Colorado in mid-May) is

MAYBE SEASONAL CHANGE

Day	High	Low	Description
Thursday	73	43	sunny
Friday	72	48	windy
Saturday	66	48	thunderstorms
Sunday	68	43	windy
Monday	70	39	sunny
Tuesday	70	43	sunny
Wednesday	66	45	partly cloudy

Such data is said to be ***discrete***. The day is listed, but not the time of day. The high temperature is forecast, but not the time of that high. The "description" is also discrete, one of the several words that are used to summarize the quality of the weather, as opposed to the quantity of rain.

Calculus is about ***continuous change***. For instance, if the weather bureau provide a web interface that let me enter the date and time to the nearest fraction of a second, they would be giving a way to track the change ***continuously***. Many physical processes are intrinsically continuous, for instance the motion (change in position) of a spacecraft or the height of the tide or the stress on a tree as a function of wind velocity.

Finding a language to describe continuous change—famously, the position of the moon or planets in their orbit, or the speed of a ball rolling down a ramp—was central to the emergence of what historians call the "Age of Enlightenment" or "modern scientific method." The first complete presentation of that language was published by Isaac Newton based on his work in

the 1660s. As you might guess, the name of the language is "calculus."

10.3 Slope

You already know pretty much everything there is to know about the straight-line function,

- **Formula**: $f(x) \equiv ax + b$. The parameters a and b are the "slope" and "intercept" respectively. (More precisely, b is the "y-intercept." But in statistics and modeling, it's just the "intercept.")
- **Reading parameters from a graph**: You learned several ways to do this which are all equivalent. Maybe the easiest is to read the y-intercept off the graph. That's b. Then choose some non-zero x_0 and read off from the graph the value of $f(x_1)$. The slope is simply

$$\frac{f(x_0) - b}{x_0}$$

  The ***y-intercept method*** is a special case of a more general method, the ***two-point method***, that you can use even if the y-intercept isn't shown on the graph. Pick two specific values of x, which we'll call x_0 and x_1. Evalate the function at these input values and compute the rise over run:

$$\text{rise over run} \equiv \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

  The rise over run is the slope of the straight line.

  The y-intercept method is exactly the same as the two-point method with $x_1 = 0$.

- **Matching a straight-line function to data**: You might not have been taught this formally, but the basic process is easy to imitate. The process is called ***line fitting*** or, in statistics and other fields, ***linear regression***.

10.4 Average rate of change

Since the slope is our standard way of representing a relationship of change, we will often use it as a way of summarizing a function. To illustrate, consider the exponential model we constructed to match the cooling-water data in Section @ref(fit-exponential):

```
water <- makeFun(60*exp(-0.0173*t) + 25 ~ t)
```

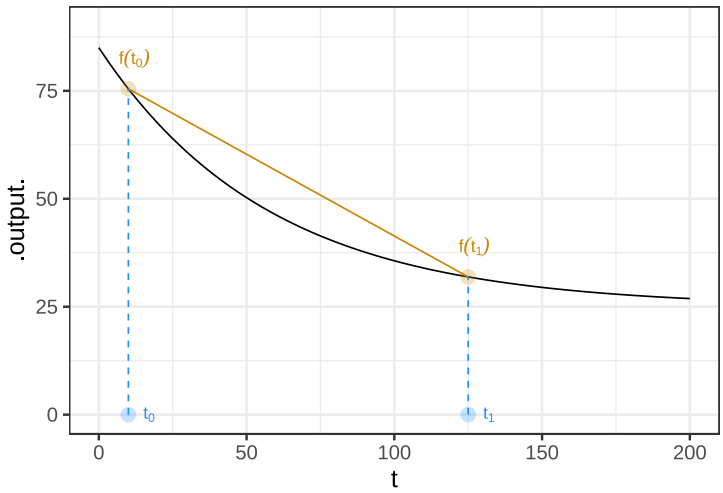


Figure 10.1: The exponential function that was previously matched to the cooling-water data. The slope of the straight line connecting two points on the function graph is the average rate of change during the interval.

During the interval $[t_0, t_1]$ the rate at which the water cools is higher at first and lower at the end. The ***average*** rate of change is a single number that summarizes the whole interval.

For all except straight-line models, the average rate of change depends on the interval chosen.

Math in the world ...

"Slope" is a natural metaphor when thinking of a function as a graph. But a more general way to describe the concept is the ***rate of change*** of the output with respect to the input. The change in the output from one end of the interval is $f(x_1) - f(x_0)$, the change in the input is $x_1 - x_0$. If the input is time (in hours), and the output is the position of a car (in miles), then the rate of change is *miles-per-hour*: the car's velocity.

For a straight-line function—think of a car driving at constant speed on a highway—it doesn't matter what you choose for x_1 and x_0 (so long as they are not identical). But for other functions, the choice does matter.

Imagine a graph of the position of a car along a road as in Figure @ref(fig:stop-and-go). Over the course of an hour, the car travelled about 25 miles. In other words, the ***average*** speed is 25 miles/hour: the *slope* of the tan line segment. Given the traffic, sometimes the car was stopped (time C), sometimes crawling (time D) and sometimes much faster than average (time B).

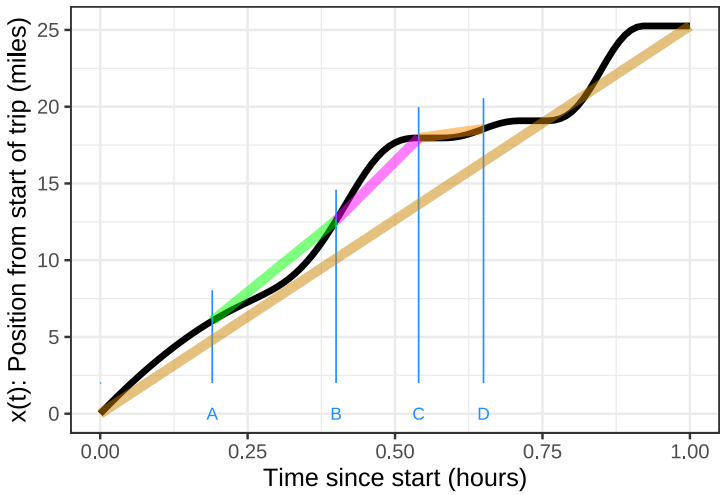


Figure 10.2: The position of an imaginary car over time (black curve). The average rate of change over various intervals is the slope of the straight-line segment connecting the start and end of the black curve in that interval.

During the interval from B to C, the car was travelling relatively fast. The slope of the magenta segment connecting the position

at times B and C is the ***average*** rate of change between times B and C. It's easy to see that the average rate of change from B to C is larger than the overall average from $t = 0$ to $t = 1$. Calculating that slope is a matter of evaluating the position at the endpoints and dividing by the length of the interval.

For instance ...

What is the average rate of change in the car's position during the interval $t_B = 0.40$ to $t_C = 0.54$?

The length of the interval is $t_C - t_B = 0.54 - 0.40 = 0.14$.

Evaluating the function gives $x(t_C) = 18$ and $x(t_B) = 12.6$.

Rise is $x(t_C) - x(t_B) = 18 - 12.6 = 5.4$.

Run is $t_C - t_B = 0.54 - 0.40 = 0.14$.

The average rate of change during the interval is $5.4/0.14 = 38.6$ miles/hour.

Math in the world ...

Figure @ref(fig:aver-tree) shows a simplified model of the amount of usable wood that can be harvested from a typical tree in a managed forest of Ponderosa Pine. (You can see some actual forestry research models here.)

You are writing a business plan for a proposed pine forest. Among other things, you have to forecast the revenue that will be generated and when you will have saleable product.

They say that "time is money." Every year you wait before harvest is another year that you don't have the money. On the other hand, every year that you wait means more wood at the end. How to decide when to harvest?

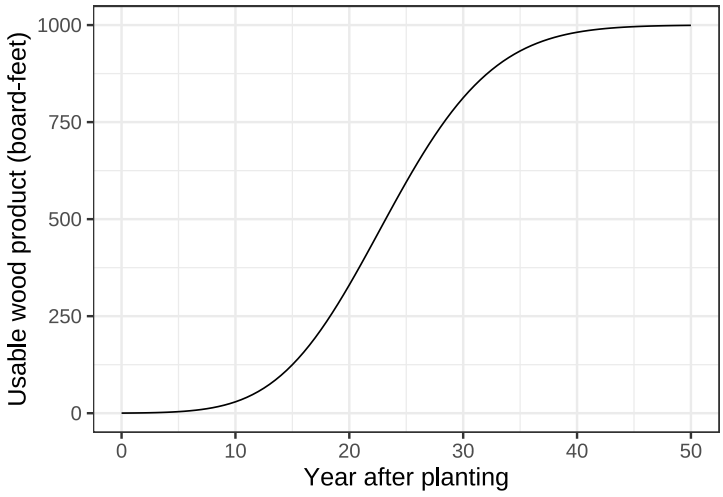


Figure 10.3: A model, somewhat realistic, of the amount of wood that can be harvested from a Ponderosa Pine as a function of years since planting to harvest.

The tree continues to grow until year 50, when it seems to have reached an equilibrium: perhaps growth goes to zero, or rot balances what growth there is. There's no point waiting until after year 50.

At year 25, the tree is growing as fast as it ever will. You'll get about 600 board-feet of lumber. Should you harvest at year 25? No! That the tree is growing so fast means that you will have a lot more wood at year 26, 27, and so on. The time to harvest is when the growth is getting smaller, so that it's not worth waiting an extra year.

The quantity of interest is the average rate of growth from seedling to harvest. Harvesting at year 25 will give a total change of 600 board feet over 25 years, giving an average rate of change of $600 \div 25 = 24$ board-feet-per-year. But if you wait until year 35, you'll have about 900 board feet, giving an average rate of change of $900 \div 35 = 25.7$ board-feet-per-year.

It's easy to construct a diagram that shows whether year 35 is best for the harvest. Recall that our fundamental model of change is the straight-line function. So we're going to ***model the model*** of tree growth as a straight line function. Like the

more realistic model, our straight-line model will start out with zero wood at the time of planting. And to be faithful to the realistic model, we'll insist that the straight-line intersect or touch the realistic model at some point in the future.

Figure @ref(fig:aver-tree2) reiterates the realistic model of the tree but adds on to it several straight-line models that all give zero harvest-able wood at planting time. Each of the green lines captures a scenario where the tree is harvested at the indicated time: t_1, t_2, and so on. For the perspective of representing the rate of growth per year from planting to harvest, the straight-line green models do not need to replicate the actual growth curve. The complexities of the curve are not relevant to the growth rate, which can be simplified down to a straight-line model connecting the output at planting time to the output at harvest time. In contrast, the magenta curve is not a suitable model because it doesn't match the situation at any harvest time; it doesn't touch the curve anywhere after planting!

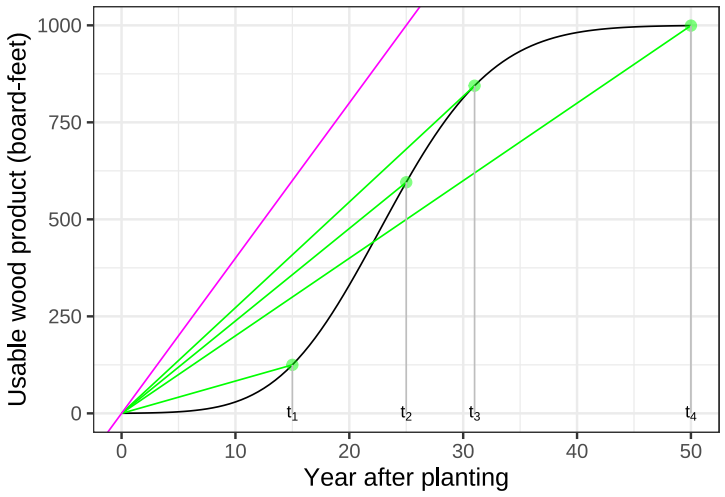


Figure 10.4: Modeling the tree-growth model with straight lines connecting planting time to various harvest times. The slope of each line is the average rate of growth for that planting time.

To maximize average lumber volume per year, choose a harvest time that produces the steepest possible green segment. From Figure @ref(fig:aver-tree2) this is the model that glances the

growth curve near year 31 (shown as t_3 in the diagram).

It's best to find the argmax by creating a function that shows explicitly what one is trying to optimize. (In Chapter @ref(optim-and-shape) we'll use the name ***objective function*** to identify such function.) Here, the objective function is ave.growth(year) ≡ volume(year)/year. See Figure @ref(fig:aver-tree3).

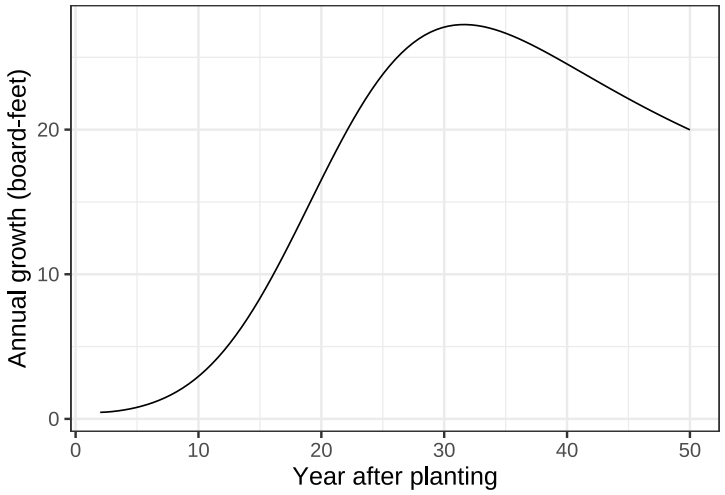


Figure 10.5: Graph of the average-growth function ave_growth(year), constructed by dividing volume(year) by year.

The graph of ave_growth(year) makes it clear that the maximum average growth from planting to harvest will occur at about year 32.

10.5 Instantaneous rate of change

The ***average rate of change*** is the slope of a line segment connecting two points on the graph of a function, the points $\big(t_0, f(t_0)\big)$ and $\big(t_1, f(t_1)\big)$. It reflects all the point-to-point changes in the value of the function over the interval t_0 to t_1 in the function's domain.

It turns out to be helpful to consider the rate of change of a function at an individual point t_0 in the domain, rather than the interval between two points. This rate of change *at a point* is called the ***instantaneous rate of change***. In Block 2, we'll see that a good way to define an instantaneous rate of change at t_0 is as the average rate of change over the interval $t_0 \le t \le t_0 + h$ with the proviso that the interval length h goes as closely as it can to zero. Visually, this is the line that's tangent to the function's graph at the input value t_0 as in Figure @ref(fig:tangent-line-exp).

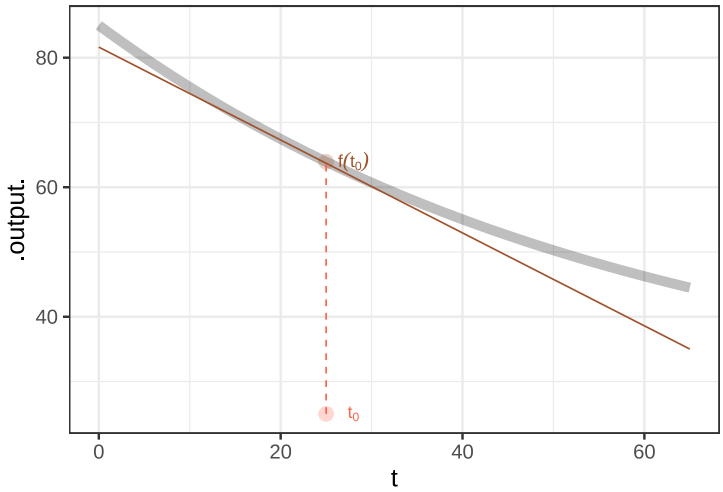


Figure 10.6: A line tangent to a the curve at a single point. The slope of this line is the instantaneous rate of change.

It's convenient to be able to find the slope of such a tangent line using just the definition $f(t)$, rather than having to draw a graph and eyeball the tangent. For now, let's approximate the slope of tangent line by the average rate of change over a small run from t_0 to $t_0 + 0.1$:

$$\text{slope of } f(t) \text{ at } t_0 \approx \frac{f(t_0 + 0.1) - f(t_0)}{0.1} = \frac{\text{amount of rise}}{\text{length of run}}$$

The $\approx$ symbol means "is approximately." For now, I want to put off the question of what "approximately" means. In modeling, whether the 0.1 gives a good enough approximation will depend on the function $f()$ and the context in which the slope

is needed. For instance, in drawing Figure @ref(fig:tangent-line-exp) I needed to find the tangent line. Using 0.1 is entirely satisfactory in this setting but it might not be in other settings.

The notation "slope of $f(t)$ at $t = t_0$" is long-winded and awkward. If we were looking at the "value of $f(t)$ at t_0 we have at hand a much more concise notation: $f(t_0)$. But it doesn't work to write"slope of $f(t_0)$" because $f(t_0)$ is a **quantity** and not a **function**. Instead, let's make a concise notation for "slope of $f(t)$." Following tradition, we'll write $\mathcal{D}f(t)$. The name of this "slope of $f(t)$" function is $\mathcal{D}f()$: a two-letter name. When we want to say, "the (approximate) slope of the tangent line to $f(t)$ at t_0, we can write simply:

$$\mathcal{D}f(t_0)$$

meaning, evaluate the"slope function of f()" at t_0.

To formalize this, we'll define the ***slope function of f()*** as

$$\mathcal{D}f(t) \equiv \frac{f(t+0.1) - f(t)}{0.1}$$

Let's look at the slope functions that correspond to some of pattern-book functions: e^x, $\sin(x)$, x^{-1} and $\ln(x)$. We can define them easily enough in R:

```
Dexp <- makeFun((exp(t+0.1) - exp(t))/0.1 ~ t)
Dsin <- makeFun((sin(t+0.1) - sin(t))/0.1 ~ t)
Dxm1 <- makeFun(((1/(t+0.1)) - (1/t))/0.1 ~ t)
Dlog <- makeFun((log(t+0.1) - log(t))/0.1 ~ t)
```

Why did you?

Why did you plot both the function and the slope function in the same graphics frame?

Excellent question! In general, it is illegitimate to plot a function and it's slope function on the same vertical axis. The

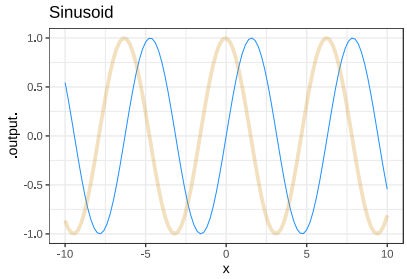


Figure 10.7: Comparing the pattern-book function (blue) to its slope function (tan)

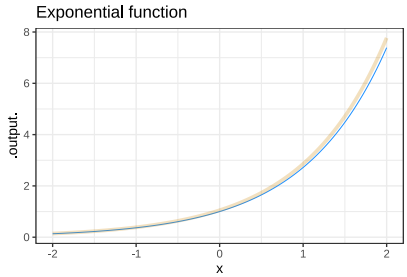


Figure 10.8: Comparing the pattern-book function (blue) to its slope function (tan)

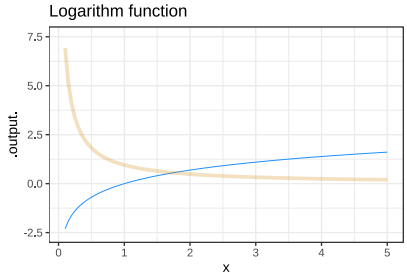


Figure 10.9: Comparing the pattern-book function (blue) to its slope function (tan)

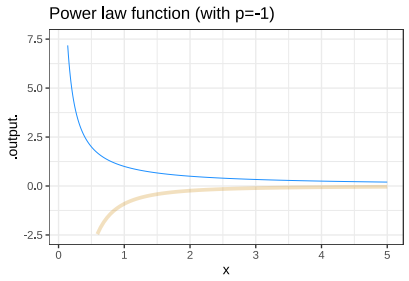


Figure 10.10: Comparing the pattern-book function (blue) to its slope function (tan)

reason is the units of the two functions will be different. For instance, the output of a function position(t) might have units of "miles," while the output of the slope function of position (that is, $\mathcal{D}$position(t) would have units such as miles-per-hour.) So, as a general rule, never plot a function and its corresponding slope function on the same scale.

An exception is for the pattern-book functions. These always take a number as input and produce a number as output. The slope function of a pattern-book function also produces a number as output.

This exception is not a good excuse for indulging a bad practice. Perhaps you'll forgive the authors if they claim they wanted to emphasize the point by demonstrating it.

Take note!

Here, we write the slope function of $f(t)$ as $\mathcal{D}f(t)$. That works for this chapter, which deals with functions of only one variable. But in general modeling functions have more than one variable, for instance $g(x, t)$. To work with slope functions with more than one variable, we need to extend the notation a little. We will place a small subscript after $\mathcal{D}$ to indicate which variable we are changing. Thus, there will be two slope functions for

$g(x,t)$:

$$\mathcal{D}_x g(x,t) \equiv \frac{g(x+0.1,t)-g(x,t)}{0.1}$$

and

$$\mathcal{D}_t g(x,t) \equiv \frac{g(x,t+0.1)-g(x,t)}{0.1}$$

The input referred to in the subscript following $\mathcal{D}$ is called the ***with-respect-to input***.

The idealization of the slope function involves replacing $h = 0.1$ with something much smaller. What "much smaller" means has been a complicated issue in the history of calculus. Today, we write $h \to 0$ to signify the process of making h smaller and smaller, but never zero. When h has this "much smaller" value, the rate of change over the interval x to $x+h$ becomes a rate of change "at x", also called the ***instantaneous rate of change*** at x. For the pattern-book functions, $h = 0.1$ or smaller gives a pretty good approximation to the instantaneous rate of change. Later, in Block 2, we'll see how to arrange h so that it's "much smaller" for functions in general.

Math in the world ...

In the previous section we looked at the optimal time to harvest a tree so that the average rate of growth in usable lumber over the tree's life is maximized. Using a model of tree growth of a ponderosa pine we found the best harvest time to be 32 years.

Let's return to the modeling phase of the wood-harvest problem with a new perspective. The real objective of tree farming is to maximize the **economic value** of the wood. This depends on the market price of the wood which itself may be changing in time. A market-savvy modeler will want to exploit any information about the possibility of rising or falling prices in selecting the best harvest time. Companies often hire economists to forecast market trends, but this requires a deep knowledge

of trends in supply and demand which is out of the scope of what we can cover in this book.

However, there is one economic principle that we can incorporate into the model without such detailed, industry specific expertise. This is the economic principle of ***opportunity cost***.

Opportunity cost takes into account when valuing an asset the other possible uses of that asset. For example, lumber companies constantly invest in planting new trees for future harvest. In order to do this, they borrow money and they pay interest on the borrowed money. They need to borrow because their existing assets are tied up in the form of wood. The opportunity cost of not harvesting a tree is the interest on the loan the company needs to take out in order to invest for the future.

Between year 30 and 32, there is hardly any change in the value of the average-rate-of-change function. It's increasing a little, but is it really worthwhile to wait? One argument is that at year 30 you already have a valuable resource: wood that could be money in the bank. If the money were in the bank, you could invest it and earn more money *and* at the same time get a new seedling in the ground to start its growth. You're doing two things at once. Efficient!

To know what is the best year for harvest from this point of view, you want to calculate the effective "interest rate" on the present amount of wood that you earn in the form of new wood. That interest rate is the ratio of the *instantaneous* rate of growth of new wood divided by the amount of existing wood. Figure @ref(fig:tree-interest) shows this function.

Early in the tree's life, the growth is high compared to the volume of the tree. That's because the tree is small. As the years pass, the tree gets bigger. Even though the rate of growth increases through year 23, the accumulated volume increases even faster, so there is a fall in the rate of return.

The best time to harvest is when the annual "interest rate" paid by the growing tree falls to the level of the next best available investment. Suppose that investment would pay 10% per year. Then harvest the tree when the function values falls below 10%. That happens at year 24. If the next best investment paid only 5% (blue horizontal line), the harvest should be made at

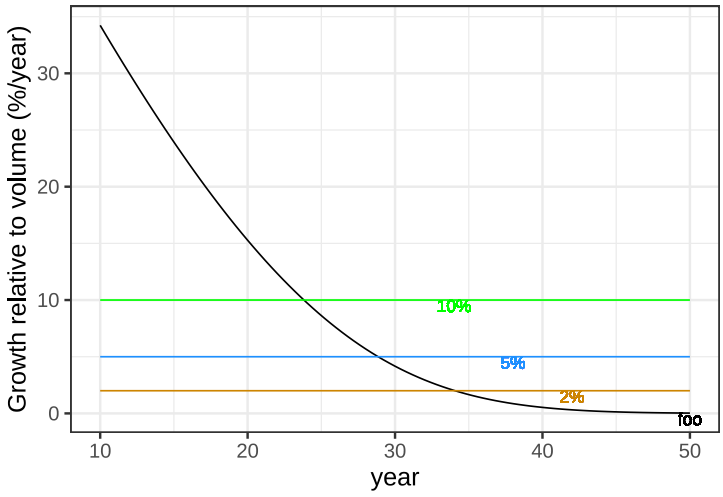


Figure 10.11: The instantaneous investment return from the tree is the instantaneous rate of change in wood volume divided by the wood volume itself. This falls over the age of the tree as the harvestable wood volume increases.

about year 29. If money could be borrowed at 2%, it would be worthwhile to harvest the tree still later.

10.6 Other old stuff

The ***rate of change*** is based on a simple question: If the input changes from $x = A$ to $x = B$, how much does the output change? Of course, the output from function $f(x)$ will be $f(x = A)$ and $f(x = B)$ respectively. The rate-of-change relationship is the ratio

$$\frac{f(x = B) - f(x = A)}{B - A} \quad \text{also written} \quad \frac{\text{rise}}{\text{run}}$$

Why do we focus on the ***rate of change*** rather than something simpler, for example the ***net change*** $f(x = B) - f(x = A)$? The reason goes back to a scientific breakthrough in the 1600s:

the writing down of the Newton's laws of motion. The language in which these laws were first successfully expressed is the language of rates of change. In the intervening 400 years, the laws have been updated with the theory of relativity and quantum mechanics. These laws too are expressed as rates of change. In undertaking to study just about any quantitative field from engineering to economics you'll find that theory is expressed using functions and rates of change.

You may recognize in the formula for the rate of change a familiar quantity: the slope of a line. Everyone understands what a line is, but the geometry is not our primary concern here. We describe relationships using functions and for us the straight-line function will be a fundamental way of expressing a relationship. Straight-line functions can be written in several ways, but we'll tend to use two predominant forms:

$$\text{line}(x) \equiv ax + b \quad \text{or} \quad \text{line}(x) \equiv a[x - x_0]$$

The two forms are interchangeable, but as you'll see in upcoming chapters, sometimes it's more convenient to use one form or the other. In either case, the rate of change is, quantitatively, the value of the parameter a.

The simple function $\text{line}(x)$, whose change relationship we understand intuitively, will be used to approximate more complicated change relationships. With the approximation in place, we can do calculations about the change relationships much more easily. Collectively, the set of mathematical concepts and techniques that support describing and calculating on change relationships has the name *Calculus*.

10.7 Exercises

Exercise 1.2: KHDUE unassigned

Exercise 9.4: WRWIX unassigned

Exercise 9.5: KEWIX unassigned

Exercise 9.6: YQCLE unassigned

Exercise 9.7: rTSX3 unassigned

Exercise 9.8: URIMX unassigned

Exercise 9.9: RDWKW unassigned

Exercise 9.10: HRTIX unassigned

Exercise 9.12: CUXLR unassigned

Exercise 9.13: MW03ay unassigned

11 Data

11.1 Outline

11.1.1 Organization of data

what's a data frame, what's a variable/column, what's a case/row

11.1.2 Functions from data

Returns a function: spliner, fitModel

11.2 Graphics layers

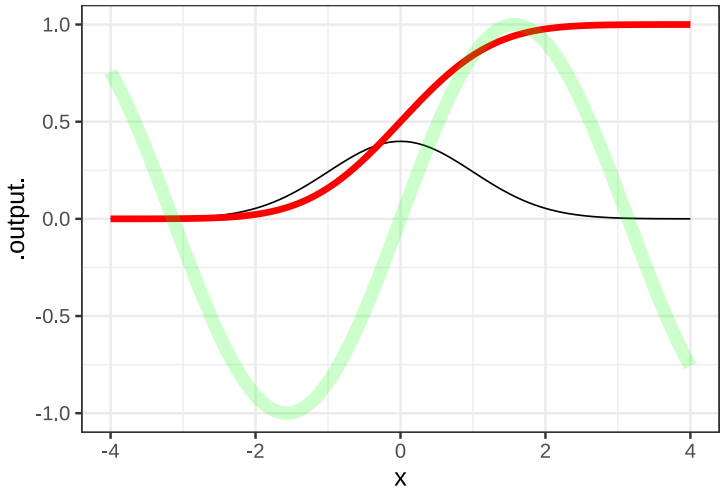
MODIFY THIS TO BE ABOUT OVERLAYING a function on data.

You will often want to compare two functions, or compare a function to data. You can do this using the ordinary graphics functions, e.g. `slice_plot()` or `gf_point()`, arranging things so that both types of graphics are drawn together in the same graphics frame. To create this kind of compound graphic, arrange the individual graphics commands into a ***pipeline***, which is a list of commands connected together by `%>%`. Your pipeline might include two commands or twenty, depending on how complicated is the graphic you want to draw. As long as you use `%>%` after each command, the next command is taken to build upon the previous command. The very last command in that pipeline should not be followed by `%>%`."

Here is an just-for-demonstration plot composed from three graphs, each displaying one of the pattern-book functions. At

the start of the pipeline, the `domain()` must be given explicitly as an argument to `slice_plot()`. You're welcome to specify other domains in the commands further along the pipeline, but if you don't the original `domain()` will be passed down the pipeline.

```
slice_plot(dnorm(x) ~ x, domain(x=c(-4,4))) %>%
  slice_plot(pnorm(x) ~ x, color="red", size=2) %>%
  slice_plot(sin(x) ~ x, color="green", size=4, alpha = 0.2)
```



Just to show how these things are done, the functions have been drawn in different colors, different widths (e.g., `size=2`) and different levels of transparency (e.g. `alpha=0.2`). You can use such styling arguments in any slice-plot.

11.3 Fitting parameters

Seen very abstractly, a mathematical model is a set of ***functions*** that represent the relationships between inputs and outputs.

At the most simple level, building a model can be a short process:

1. Develop an understanding of the relationship you want to model. Often, part of this "understanding" is the pattern seen in data.
2. Choose a function type—e.g. exponential, sinusoidal, sigmoid—that you think would be a good match to the relationship.
3. Find ***parameters*** that scales your function to be able to accept real-world inputs and generate real-world outputs.

It's important to distinguish between two basic types of model:

1. ***Empirical models*** which are rooted in ***observation*** and ***data***.
2. ***Mechanistic models*** such as those created by applying fundamental laws of physics, chemistry, and such.

We are going to put off mechanistic models for a while, for two reasons. First, the "fundamental laws of physics, chemistry, and such" are often expressed with the concepts and methods of calculus. We are heading there, but at this point you don't yet know the core concepts and methods of calculus. Second, most students don't make a careful study of the "fundamental laws of physics, chemistry, and such" until *after* they have studied calculus. So examples of mechanistic models will be a bit hollow at this point.

We'll start then with empirical modeling: finding functions that are a good summary of data. The process of constructing a model that is a good match for data is called ***curve fitting***, or, more generally, ***fitting a model***.

11.4 Variations from scaling

A good place to start building a model is to pick one of the basic modeling functions. This works surprisingly often. To remind you, here are our nine ***pattern-book*** functions:

Pattern name	Traditional notation	R notation
exponential	e^x	`exp(x)`

Pattern name	Traditional notation	R notation
logarithm ("natural log")	$\ln(x)$	`log(x)`
sinusoid	$\sin(x)$	`sin(x)`
square	x^2	`x^2`
proportional	x	`x`
one	1	`1`
reciprocal	$1/x$ or x^{-1}	`1/x`
gaussian	dnorm(x)	`dnorm(x)`
sigmoid	pnorm(x)	`pnorm(x)`

The basic modeling functions are the same, but replace the plain x in the pattern-book function with a straight-line function, for instance $ax + b$ or, equivalently $a(x - x_0)$. In use, the parameter a is often written with some other letter and, often, the b or $-x_0$ part is not needed.

Here are some of the common forms of the basic modeling functions you will encounter:

Name	Common forms	note
Exponential	e^{kt} or e^{-kt} or $e^{-t/\tau}$	
Sinusoid	$\sin\left(\frac{2\pi}{P}(t - t_0)\right)$, $\sin\left(\frac{2\pi}{P}t\right)$, or $\sin(\omega t)$	P is "period", ω is "angular frequency."
Monomials	$[x - x_0]$ or $[x - x_0]^2$, and so on	
Power-law generally	x^p or $[x - x_0]^p$	p is "power."
Gaussian	dnorm$(x, mean, sd)$	Interpret "mean" as "center."
Sigmoid	dnorm$(x, mean, sd)$	sd is "standard deviation" or "spread."

It helps in making the selection to have ready to mind the basic shape of each of these function families. To review, revisit Section @ref(function-shapes).

Remember also that, in general, we scale the inputs and scale

the output. This means that in choosing a model family, we don't have to worry at first about the numbers on the axes. (Of course, those numbers will be critically important later on in the process!) The scaling does, however, allow us to consider some variations on the shapes of the modeling functions. In particular, the ***input scaling*** lets us flip the shape right-for-left. And the ***output scaling*** lets us flip the shape top-for-bottom.

- $f(t)$, basic shape
- $f(-t)$, flipped right-for-left
- $-f(t)$, flipped top-for-bottom
- $-f(-t)$, flipped both top-for-bottom and right-for-left

For functions such as the sinusoid, flipping is not much use, since the flipped sinusoid curve is still a sinusoid, although with a shifted input. Similarly, a right-for-left flipped gaussian function has the same shape as the original. For the straight-line function, flipping of either sort accomplishes the same thing: changing the sign of the slope.

For the exponential function, the two possible types of flipping—right-for-left and top-for-bottom—produce four different curves, all of which are exponential, shown in Figure @ref(fig:four-variations).

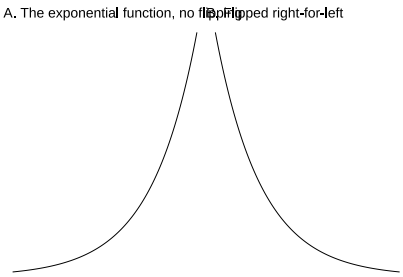


Figure 11.1: Four variations of the exponential function.

11.5 Fitting a straight-line function

To illustrate line fitting, let's return to the cooling mug of water. Figure @ref(fig:Fun-4-a) shows the data along with a dozen

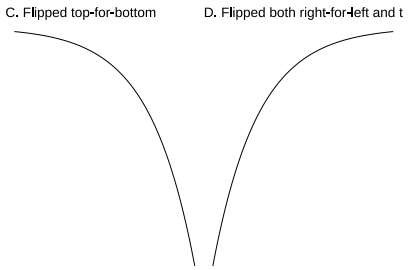


Figure 11.2: Four variations of the exponential function.

candidate straight line functions, each one drawn in its own color.

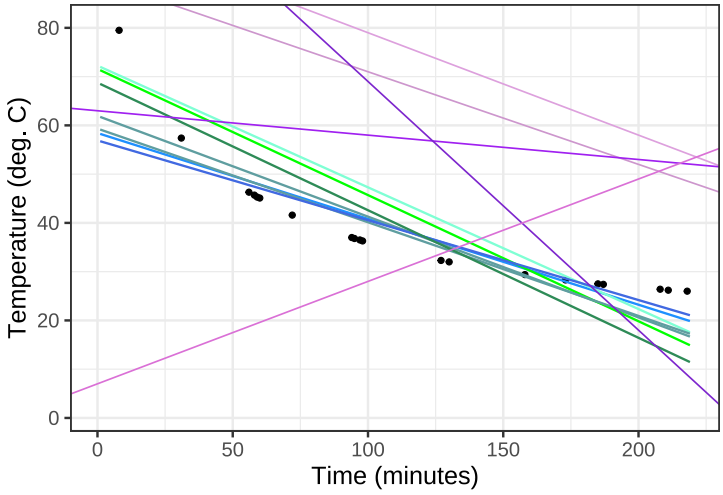


Figure 11.3: Some candidate straight-line function models plotted on top of the cooling water data. Which one(s) would you pick as good matches to the data?

Some of the straight-line models are a much better match to the data than others. The blue-shaded functions are pretty good fits, at least when you consider the limitations of matching data with a curved pattern by a straight line. The green-colored functions are maybe OK but not as good as the blue, and the purple-shaded functions are just horrible.

Now that you know what a reasonable straight-line model looks like, you will find it pretty easy to draw one on data graphics that even remotely show a straight-line pattern.

Step 1: Draw a reasonable straight-line through the data points.

Step 2: Find the parameters that correspond to the line you drew.

11.6 Curve fitting a periodic function

Figure @ref(fig:Fun-4-a-3-1) shows the tide level in Providence, Rhode Island, starting at midnight on April 1, 2020 and recorded every minute for four and a half days. (These data were collected by the US National Oceanic and Atmospheric Administration. Link)

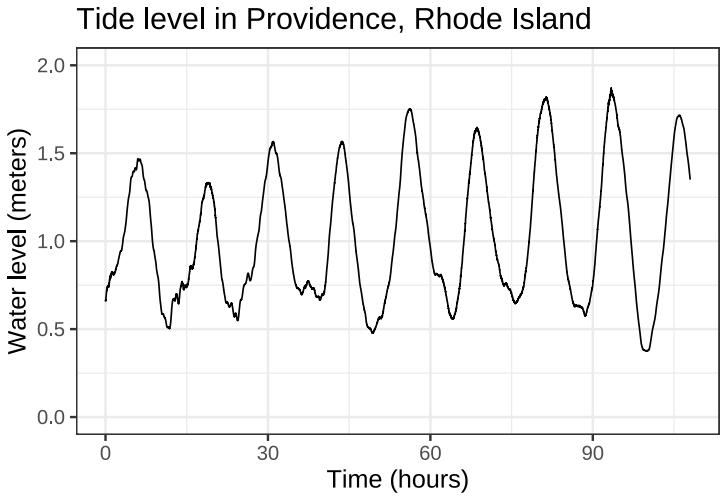


Figure 11.4: About four days of tide-level data from Providence, Rhode Island

It's not too hard to see what's going on. The tide rises and falls about every 12 hours. The difference between high tide and low tide is a little more than one meter. The tide gauge is calibrated so that a typical reading is 1 meter, although we don't know what that is respect to. (Certainly not sea level, since then the typical reading would be around zero.)

This simple description tells almost everything needed to match a basic modeling function to the tide level. Given the clear

pattern in the data, we'll use a sinusoid, that is, a function of the form

$$\text{tide}(t) \equiv A \sin(2\pi t/P) + B$$

The procedure is straightforward:

Step 0: Determine whether a sinusoid model is appropriate. As you know, sinusoids oscillate up and down repeatedly with a steady ***period***. That certainly seems the case here. But sinusoids are also steady in the ***peak*** and ***trough*** values for each cycle. That's only approximately true in the Providence data. Models inevitably involve approximation. We'll have to keep an eye on whether modeling with sinusoids and their ***fixed amplitude*** still lets us extract useful information.

Step 1: Choose a sensible value to represent the low point repeatedly reached. 0.5 m seems appropriate here, but obviously the exact position of the trough of each cycle varies over the 4.5 day duration of the data. Similarly, the peak is near 1.6 m. Parameter B is the mean of the peak and trough values: $\frac{1.6+0.5}{2} = 1.05$ m here. Parameter A is half the difference between the peak and trough values: $\frac{1.6-0.5}{2} = 0.55$. Parameter B is called the ***baseline*** of the sinusoid. Parameter A is the ***amplitude***. (Note that by convention, the amplitude is always *half* the high-to-low range of the sinusoid.)

Step 2: Estimate the period P of the sinusoid. This can be done with the horizontal axis scale: measure the horizontal duration of a complete cycle. I like to use the time between peaks, but the time between troughs would work just as well. Another good choice is the time between positive sloping crossings of the baseline. (But be careful. The time between *successive* baseline crossings, one positive sloping and the other negative, give just *half* the period.)

On the scale of the above plot, it's hard to read off the time of the first peak. So, zoom in until it becomes more obvious.

The left panel in Figure @ref(fig:Fun-4-a-3-2) shows about 24 hours at the start of the record. The first peak is at about 6 hours, the second at about 19 hours. That indicates that the period is roughly 19 - 6 = 13 hours.

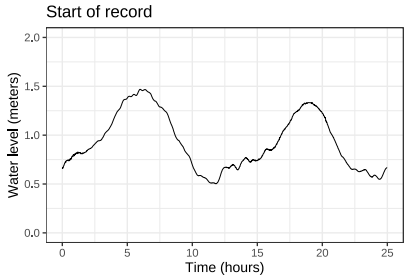


Figure 11.5: Zooming in on the start of the data (left) and on the last part of the data (right).

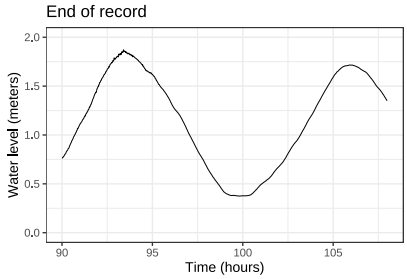


Figure 11.6: Zooming in on the start of the data (left) and on the last part of the data (right).

Step 3 Plot out the model over the data. Replacing the symbols A, B, and P with our estimates, the model is

$$\text{tide}(t) \equiv \underbrace{1.05}_{A} + \underbrace{0.55}_{B} \sin(\ 2\pi t\ /\ \underbrace{13}_{P})$$

Figure @ref(fig:Fun-4-a-3-3) shows this model in magenta.

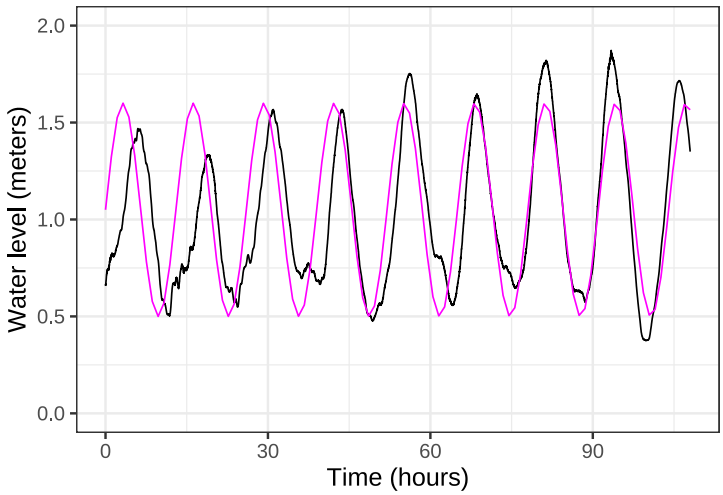


Figure 11.7: The sinusoid fails to align with the timing of peaks and troughs.

Figure @ref(fig:Fun-4-a-3-3) shows the model aligning beautifully with the data at around time 80 hours, but not so well near the very beginning of the record. Looking carefully, you can see the magenta peak gradually move to the left compared to the data as you look at the peaks of the cycles one at a time moving backward from $t = 80$ hours. This is diagnostic of our 13-hour estimate for the period being a little too long.

A good way to refine the estimate is to change the model slightly and re-graph the data and model. Keep doing this until you have found the right alignment. In other words, parameter estimation is often an ***iterative*** process of estimate-and-refine. This is one of the aspects of the ***modeling cycle***, where a modeler builds a tentative model, looks for systematic deviations from the data, and refines the model.

Another sort of deviation of the model from the data seen in Figure @ref(fig:Fun-4-a-3-3) concerns the varying heights of the peaks and troughs in the data, which is not captured by the simple sinusoid pattern. Perhaps you can see that at the beginning of the record, the troughs are also *wider* than the peaks. Later on, as this extra width disappears, the amplitude of the peaks and troughs increases.

You don't yet have the calculus tools or experience to know how or whether this phenomenon can be modeled. For now, a hint: Earth has two large orbiting bodies, the Sun and the Moon. These have slightly different periods: 24 hours for the Sun and slightly longer for the Moon.

Step 4 Our model omitted one of the parameters of the sinusoid basic modeling function: the time shift t_0. A more complete model would be:

$$\text{tide}_{\text{shifted}}(t) \equiv A\sin(2\pi(t - t_0)/P) + B$$

Whether including this parameter is important depends on our purpose for the model. If the goal is to find the period of the tides, the time shift hardly matters. But if the goal is to predict a future tide level, the time shift is critical.

Estimating t_0 can only be done once the period P is known precisely. In practice, as you'll see in Chapter @ref(modeling-cycle), we use a numerical method called ***polishing*** to estimate both P and t_0 at the same time.

11.7 Curve fitting an exponential function

The exponential function is particularly useful when the quantity we want to model shows ***constant proportional increase***. Many quantities in everyday life are this way. For instance, an increase in salary is typically presented in a format like "a 3% increase." The population growth of a country is often presented as "percent per year." Inflation in the price of goods is

similarly described in percent per year. Interest on money in a bank savings account is also described as percent per year. But if you have the bad fortune to owe money to a loan shark, the proportional increase might be described as "percent per month" or "percent per week."

When you know the "percent increase per time" of a quantity whose initial value is A, the exponential function is easy to write down:

$$f(t) = A(1 + r)^t$$

The number r is often called the ***interest rate*** or ***discount rate*** and is given in **percent**.

TAKE NOTE!

Regrettably, it's extremely easy and common to forget the rules for addition with percent. If $r = 5\%$, then $(1 + r) = 1.05$, not 6. Always keep in mind that 5% means $\frac{5}{100}$.

Another source of error stems from the tradition in mathematics of using the number $e = 2.718282$ as the "natural" base of the exponential function, whereas in $f(t) = A(1 + r)^t$ the base is $1 + r$.

You can translate an exponential b^t in any base b to the "natural" base. This is just a matter of calculating the appropriate parameter k such that $e^k = b$. Using logarithms,

$$e^k = b \quad \Longrightarrow \quad k = \ln(b)$$

For instance, if $r = 5\%$ per year, we'll have $k = \ln(1 + r) = \ln(1.05) = 0.488$ per year.

Almost everybody is happier doing arithmetic with numbers like 2 and 10 rather than $e = 2.718282$. For this reason, you may see formulations of the exponential function as $g(t) \equiv 2^{at}$ or $h(t) \equiv 10^{ct}$. Remember that 2^{at} and e^{at}, although both exponential functions, are quantitatively different. If you want to write 2^{at} using the "natural" base, it will be $e^{\ln(2)a\,t}$. Similarly, $10^{ct} = e^{\ln(10)c\,t}$.

Exponential functions also describe ***decrease*** or ***decay***. Just replace t with $-t$. That is, a movie of a decreasing quantity is just the movie of an increasing quantity played backwards in time!

Figure @ref(fig:Fun-4-intro-1) shows some data collected by Prof. Stan Wagon to support his making a detailed mechanistic model of an everyday phenomenon: The cooling of a mug of hot beverage to room temperature. The mug started at room temperature, which was measured at 26 degrees C. At time 0 he poured in boiling water from a kettle and measured the temperature of the water over the next few hours.

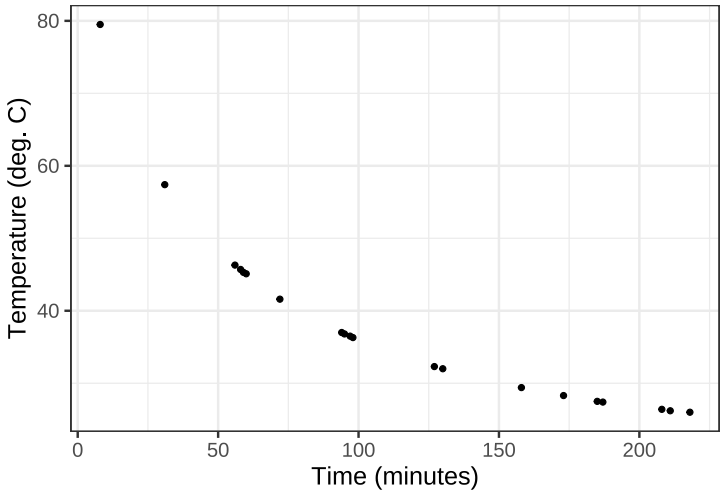


Figure 11.8: Stan's data

Our task is to translate this data into the form of a function that takes time as input and returns temperature as output. Such a model would be useful for, say, filling in the gaps of the data. For instance, we might want to find the temperature of the water immediately after being poured from the kettle into the mug.

Looking at the data, one sees that the temperature decreases along a curve: cooling fast at first and then more slowly. This is the pattern of the flipped right-for-left exponential. (Figure @ref(fig:four-variations)(B)) We can imagine then that an exponential, $Ae^{kt} + C$ will be a suitable model form for the cooling water.

What remains is to find the parameters A, k, and C. Here is a general process for curve-fitting an exponential. Later, we'll apply this process specifically to the water-cooling situation.

General process for curve-fitting an exponential

The goal is to find the parameters A, k, and C in the formula $Ae^{kx} + C$.

Step 0: Check that the data show an exponential pattern in one of its varieties, namely a smooth increase or decrease and leveling out beyond some value of x. If this isn't true, reconsider whether you should be using an exponential function in the first place. Using your mind's eye (or paper, if you like) sketch out an exponential-shaped curve that follows the overall trend of the data. We'll call this imagined curve $f(x)$.

Step 1: Find the ***baseline***. This is the output level at which the function has a horizontal asymptote, that is, at which the function levels off. C is this baseline level.

Step 2: Find the numerical value of the imagined function $f()$ at input $x = 0$. We'll call this value $f(0)$. Then $A \equiv f(0) - C$.

Step 3 Estimate parameter k using these steps:

a. Pick any input value, which we'll call x_2, such that $f(x_2)$ is far from the baseline.

b. Find an input value, which we'll call x_1 such that $f(x_1)$ is halfway[1] between the baseline C and $f(x_2)$, that is

$$f(x_1) = \frac{f(x_2) + C}{2}$$

[1]The motivation for the equation is that the distance between $f(x_1)$ and C should be half that of the distance between $f(x_2)$ and C. In other words,

$$f(x_1) - C = \frac{f(x_2) - C}{2}$$

Simplifying the algebra a little tells you to look for x_1 such that

$$f(x_1) = \frac{f(x_2) + C}{2}$$

c. Once you have x_1 and x_2, you can immediately find k:

$$k = \frac{\ln(2)}{x_2 - x_1} \approx \frac{0.693}{x_2 - x_1}$$

d. Using this simple test to double-check your work. If the horizontal asymptote of $f()$ (that is, approach to the baseline) is for $x \to \infty$, then k should be **negative**. If the horizontal asymptote is for $x \to -\infty$, then k will be positive.

Notice that the question of "growth or decay" depends only on the sign of the parameter k. You can have an exponentially decaying process that's increasing. Consider, for instance, the speed of a car as it merges onto a highway from a slow speed on the entrance ramp. The car's velocity is increasing, but as you approach highway speed the rate of increase gets smaller. That's exponential **decay**.

The procedure in Step 3 for estimating k stems from a very important property of exponential functions: Exponential functions always double/half at a constant pace. By design, the parameter k directly determines that pace. Picking an x_2 and finding the corresponding x_1 gives the length of the input interval, $x_1 - x_2$ over which the distance from the baseline doubles/halves.

What's remarkable about the doubling/halving time is that, for a genuinely exponential function, it doesn't matter which point we choose for x_1. In practice working with graphed data, it's best to choose so that $f(x_1)$ is discernibly far from the baseline.

The 2 in $\ln(2)$ corresponds to idea of doubling/halving. The logarithm converts 2 to a scale that will generate 2 when exponentiated.

Step 4 If you can plot the data, do so. Add to that a graphics layer showing the function $Ae^{kx} + C$ using the values for A, C, and k that you just found. If you are not satisfied with the way the plotted function approximates the data, tweak the parameters a bit until you are.

Exponential curve fitting applied to the water-cooling data

Let's illustrate the general process on the water-cooling data, redrawn in Figure @ref(fig:Fun-4-a-2-1).

```
## `geom_smooth()` using method = 'loess'
```

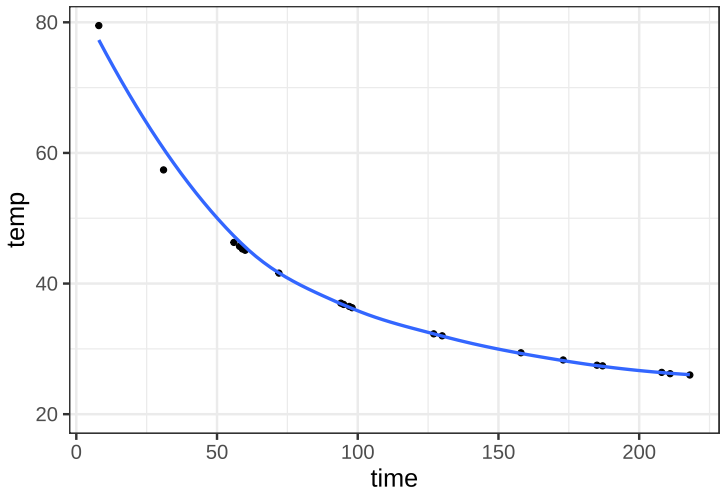


Figure 11.9: The cooling-water data, repeated here for convenience. We've sketched in an exponential-like curve that matches the data pretty well.

Step 0: The data indicate a smooth curve. As t gets large, the curve approaches a constant. So an exponential model is reasonable.

Step 1: The curve looks like it's leveling out at a temperature of about 25 degrees C for large t. So $C \approx 25°\text{C}$.

Step 2: Looking at our imagined curve (sketched in blue in Figure @ref(fig:Fun-4-a-2-1)), it appears to intersect the $t = 0$ vertical axis at about $f(0) = 85°\text{C}$. Thus,

$$A = 85°\text{C} - 25°\text{C} = 60°\text{C}$$

Step 3:

a. Now choose a time t_2 where $f(t_2)$ is far from the baseline. … It looks like $t_2 = 25$ will do the job, at which point we can read off the graph that the function value is $f(25) \approx 65$.
b. Find an input value t_1 such that $f(t_1) - C = (f(t_2) - C)/2$. This tells us that $f(t_1) = 25 + (65 - 25)/2 = (65 + 25)/2 = 45$. Referring to the graph, the time at which the function is about 45 is $t_1 \approx 65$, that is, $f(t_1) \approx 65$.
c. We have $t_1 \approx 65$ and $t_2 = 25$. From this, we calculate k:

$$k = \frac{\ln(2)}{25 - 65} = 0.693/(-40) = -0.0173$$

d. The data show exponential decay, so we expect k to be negative. Happily, it is. If it hadn't been, we would go back to look for a sign error someplace.

Step 4. Graph the function on top of the data to confirm the fitted function is a good match to the data.

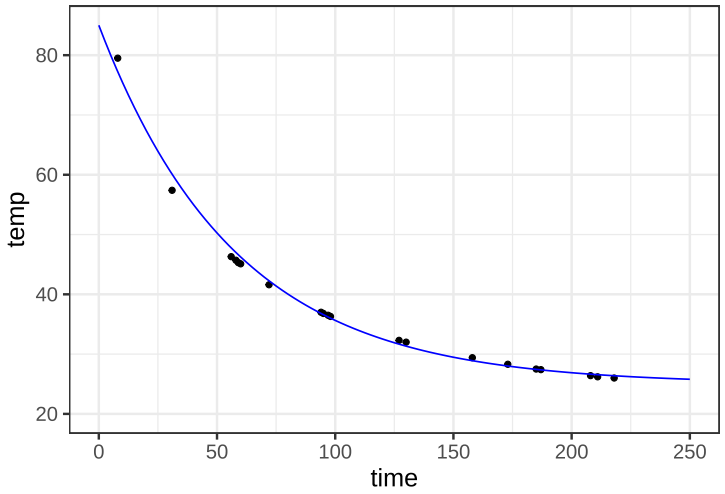


Figure 11.10: An exponential function that roughly aligns with the data.

Step 5: The flat zone of the data is to the right. So we've got exponential decay and $k < 0$.

11.8 Curve fitting a power-law function

You have been using power-law functions from early in your math and science education. Some examples:

Setting	Function formula	exponent
Circumference of a circle	$C(r) = 2\pi r$	1
Area of a circle	$A(r) = \pi r^2$	2
Volume of a sphere	$V(r) = \frac{4}{3}\pi r^3$	3
Distance traveled by a falling object	$d(t) = \frac{1}{2}gt^2$	2
Gas pressure versus volume	$P(V) = \frac{nRT}{V}$	-1
… perhaps less familiar …		
Distance traveled by a diffusing gas	$X(t) = D\sqrt{t}$	1/2
Animal lifespan (in the wild) versus body mass	$L(M) = aM^{0.25}$	0.25
Blood flow versus body mass	$F(M) = bM^{0.75}$	0.75

One reason why power-law functions are so important in science has to do with the logic of physical quantities such as length, mass, time, area, volume, force, power, and so on. We'll discuss this at length later in the course and the principles will appear

throughout calculus.

As for finding the power law $f(x) \equiv Ax^p$ that provides a good match to data, we'll need some additional tools to be introduced in Chapter @ref(magnitudes).

11.9 Gaussian and sigmoid functions

Our last two basic modeling functions express an important idea in modeling: ***localness***. To put this in concrete terms, imagine creating a function to depict the elevation above sea level of a long road as a function of distance in miles, x, from the start of the road. If the road were level at 1200 feet elevation, a sensible model would be elevation$(x) = 1200$ft. If the road were gently sloping, a better model would be elevation$(x) = 1200 + 0.01x$.

Now let's add a bump to the road. A bump is a local feature, often only a few feet wide. Or, perhaps the road is crossing a mountain range. That's also a local feature, but unlike a bump in the road a mountain range extends for many miles.

The basic modeling function suited to represent bumps in the road, or potholes, or mountain ranges is generically called a "hump function." In this book, we use a specific hump function called the ***gaussian*** function (`dnorm()`).

A gaussian function has two parameters: the location[2] of the peak, which we'll call the ***center*** parameter, and the sideways extent of the gaussian, which is called the ***standard deviation***. Figure @ref(fig:Fun-3C-1) shows a few gaussian functions with different parameters.

It's easy to read off the ***center*** parameter from a graph of a gaussian. It's the location of the top of the function graph. (We mentioned before that a mathematical word for "the location of the top" is ***argmax***; the value for the input of the function that produces the maximum output.)

The ***spread*** parameter is also pretty straightforward, but you first have to become familiar with an unusual feature of the

[2]That is, the input value at which the function value is largest.

Figure 11.11: Gaussians with various centers and standard deviations (sd).

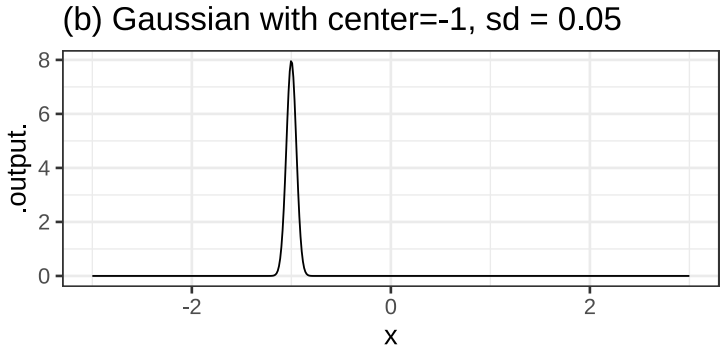


Figure 11.12: Gaussians with various centers and standard deviations (sd).

Figure 11.13: Gaussians with various centers and standard deviations (sd).

gaussian function. The output of the gaussian *far from the center* is practically zero. But it is not exactly zero. You can see from the graphs that the gaussian function has long flanks which approach zero output more or less in the manner of an exponential function. This means that we can't measure the spread of the gaussian function by the distance between the zeros on either side of the peak. Instead, we need a ***convention*** that will allow us to be precise in quantifying what is admittedly a vague concept of width.

Technically, the convention is that the spread is the length of the interval from the argmax to the inflection point. This can be hard to judge from a graph, but a reasonable approximation is that the spread is the "half-width at half-height." Come down half-way from the peak value of the output. Panel (c) of Figure @ref(fig:Fun-3C-1) marks that elevation with a thin, tan, horizontal line. Along that line, measure the width of the gaussian, as marked by the thick tan line in Panel (c). The ***spread*** parameter is half the width of the gaussian measured in this way.

If you have a keen eye, you'll notice that the tan line in Figure @ref(fig:Fun-3C-1) is not exactly half-way down from the peak. It's down 39.35% from the peak. This corresponds exactly to the technical convention.

Another seeming oddity about the gaussian function is the value of the maximum. It would have seemed natural to define this as 1, so-called "unit height." The way it actually works is different: the maximum height is set so that the ***area*** under the gaussian function is 1.

This business with the area will make more sense when you've learned some calculus tools, specifically "differentation" and "integration." For now though …

Consider another road feature, a local ***change*** from one elevation to another as you might accomplish with a ramp. The basic modeling function corresponding to a ***local change*** from one level to another is the ***sigmoid*** function. Figure @ref(fig:Fun-3C-2) shows three sigmoid functions.

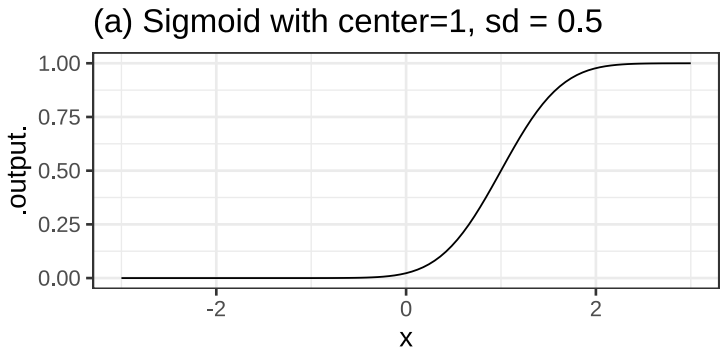


Figure 11.14: Sigmoids with various centers and standard deviations

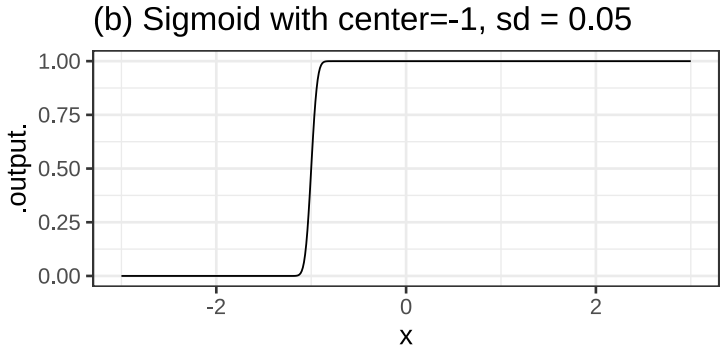


Figure 11.15: Sigmoids with various centers and standard deviations

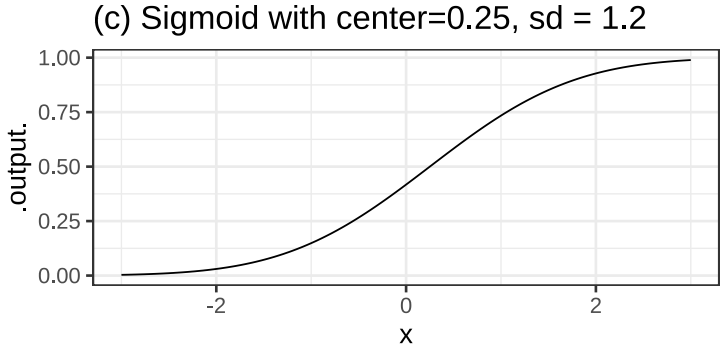


Figure 11.16: Sigmoids with various centers and standard deviations

The name "sigmoid" comes from vague resemblance of the graph to the letter S (which is "sigma" in Greek: ς).

The parameters of the sigmoid function are the same as for the gaussian function: ***center*** and ***width***. The center is easy to estimate from a graph. It's the value of the input that produces an output of 0.5, half-way between the max and min of the sigmoid. As with the gaussian function, the ***width*** is measured according to a convention. The width is the ***change in input*** needed to go from an output of 0.5 to an output of 0.8413. This use of 0.8413 must seem loony at first exposure, but there is a reason. We'll need more calculus tools before it can make sense.

Gaussian functions and sigmoid functions with the same center and width parameters have a very close relationship. The ***instantaneous rate of change*** of the sigmoid function is the corresponding ***gaussian*** function. Figures @ref(fig:Fun-3C-1) and @ref(fig:Fun-3C-2) show corresponding gaussian and sigmoid functions. To the very far left, the sigmoid function is effectively flat: a slope near zero. Moving toward the center the sigmoid has a gentle slope: a low number. In the center, the sigmoid is steepest: a higher number. Then the slope of the sigmoid becomes gentle again before gradually falling off to zero. Near zero, then low, then higher, then low again, then falling off to zero: that's also the description of a gaussian function!

In R, the name of the sigmoid function is `pnorm()`. The gaussian is `dnorm()`. The parameters that specify center and spread are named `mean` and `sd`. The word "mean" accurately conveys the idea of "center." It would be nice to be able to say that `sd` comes from `spread`, but in fact `sd` is short for ***standard deviation***, a term familiar to students of statistics.

Math in the world ...

Figure @ref(fig:ebola-data) shows the cumulative number of Ebola cases during an outbreak in Sierra Leone from May 1, 2014 to December 16, 2015.

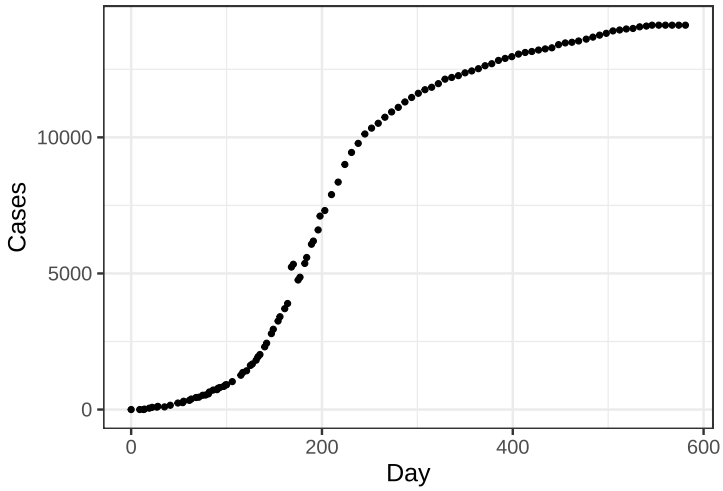


Figure 11.17: Cumulative Ebola cases in Sierra Leone

Although the cumulative case data are roughly sigmoidal in shape, there are systematic differences in shape from a true sigmoid. For comparison, Figure @ref(fig:true-sigmoid) is a graph of genuinely sigmoidal data.

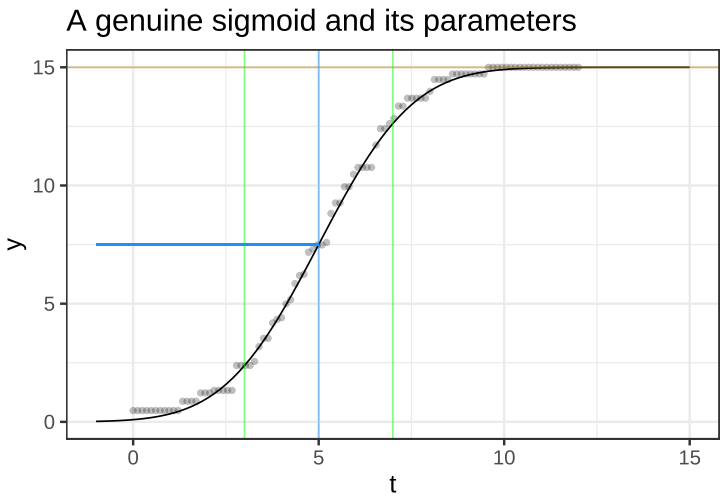


Figure 11.18: A simulated sigmoidal growth process.

The Ebola data have only a rough similarity to the sigmoid shape. Still, fitting a model and then examining closely the deviations of the model from the data can prompt questions

that can lead to a better understanding of the data and what's needed in an appropriate model.

Here's a methodology for estimating the parameters `mean` and `sd` of a sigmoid graphically.

1. Sketch in a S-shaped curve that smoothly follows the data. In Figure @ref(fig:true-sigmoid) this has already been done for you. For the Ebola data, you will have to use your judgment.
2. Find the top plateau of the S-curve. This is indicated by the tan line in Figure @ref(fig:true-sigmoid). The parameter `A` is simply the height of the plateau, in this case $y \approx 15$.
3. Come down half way from the plateau. Here, that's 15/2 or 7.5, indicated by the horizontal blue line segment. Find the inverse of the S-curve from that half-way point onto the horizontal-axis. Here, that gives $t \approx 5$. The parameter `center` is that value.
4. From the center of the S-shaped curve, follow the curve upward about 2/3 of the way to the plateau. In the diagram, that point is marked with a green line at $t \approx 7$. The `width` is the distance along the horizontal axis from the blue centerline to the green line. Here, that's $7 - 5$ giving 2 as the `width`.
5. You might also want to trace the S-curve *downward* from the centerline about 2/3 of the way to zero. That's indicated by the left green line. In the standard sigmoid, the two green lines will be equally spaced around the centerline. Of course the data may not be in the shape of the standard sigmoid, so you might find the two green lines are not equally spaced from the center.

11.10 Exercises

```
rr insert_calcZ_exercise("8.1", "0rvpbu", "Exercises/kid-type-boat.Rmd")
```

```
rr insert_calcZ_exercise("8.3", "YLWP1", "Exercises/flipping-1.Rmd")
rr insert_calcZ_exercise(8.5, "YELXG", "Exercises/lamb-talk-gloves.Rmd")
rr insert_calcZ_exercise(8.7, "VBWD", "Exercises/spirometer.Rmd")
rr insert_calcZ_exercise(8.9, "vkwl4", "Exercises/chicken-choose-vase.Rmd")
rr insert_calcZ_exercise(8.11, "asevss", "Exercises/cow-type-kayak.Rmd")
rr insert_calcZ_exercise(8.13, "IELWV", "Exercises/hump-intro.Rmd")
rr insert_calcZ_exercise(8.15, "CKSLE", "Exercises/sigmoid-intro.Rmd")
rr insert_calcZ_exercise(8.17, "bllKR", "Exercises/sigmoid-bath.Rmd")
rr insert_calcZ_exercise(8.19, "YLWP2", "Exercises/flipping-2.Rmd")
rr insert_calcZ_exercise(8.21, "EKCIE", "Exercises/ebola-sigmoid.Rmd")
```

12 Low order polynomials

12.1 The modeling polynomial

Sometimes, in order to model some simple relationship you need to build a function whose graph has a simple, curving shape.

A simple but surprisingly powerful approach is to use a ***low-order polynomial***. The ***order of a polynomial*** is the highest exponent on the input. For example:

- A ***straight-line function***, $g_1(x) \equiv a_0 + a_1 x$, is a **first-order** polynomial.
- A ***quadratic***, $g_2(x) \equiv b_0 + b_1 x + b_2 x^2$ is a **second-order** polynomial.

Many modelers are tempted to extend the technique to third-, fourth-, fifth-order and even higher. This is only rarely worthwhile since all second-, fourth-, sixth- and higher-even-order ***monomials*** have basically the same U-shape, like a referee signalling a touch-down. Similarly, first-, third-, fifth- and higher odd-order monomial have similar shapes.

An ofttimes better approach is to compose the polynomial with a curved but monotonic function, such as a logarithm.

Why did you?

Notice that in writing low-order polynomials like

$$g_1(x) \equiv a_0 + a_1 x$$

or

$$g_2(x) \equiv b_0 + b_1 x + b_2 x^2$$

we are using a specific naming convention for the scalars in the linear combinations. For each different function, we use a different start-of-the-alphabet name, like a and b. That same name is used for all the scalars in the function, and a subscript is used to make the distinction between the different functions being combined. Thus, we have a_1 for the x function in $g_1()$ and b_2 for the x^2 function in $g_2()$.

In high-school mathematics, polynomials are often written without subscript, for instance $ax^2 + bx + c$. This can be fine when working with only one polynomial at a time, but in modeling we often need to compare multiple, related polynomials.

12.2 Parentheses

WHY DID YOU?

In writing $\text{line}(x) \equiv a[x - x_0]$ I used ***square braces*** [ ] rather than parentheses ( ) to surround $x - x_0$. Either could be used and there is no difference in meaning. In traditional mathematical notation, either serves to demarcate a sub-expression. We'll be using parentheses very extensively in expressions like $\text{line}(x)$, so it's nice to have a visual break.

REVIEW STRAIGHT-LINE FUNCTION and zero finding of it.

One of the central techniques in calculus is to use straight-line functions to approximate more complicated functions. We'll start using this technique extensively in Block 2, but it's worth

pointing out some basic symbolic manipulations of the straight-line function.

1. Find the zero crossing (which you might have called "x-intercept")
2. Find the slope.
3. Point/slope form. $g(x) \equiv m(x - x_0)$ which implies $b = -mx_0$

NOTE NOTE NOTE

Much of this material is redundant with Block 2 chapter 25. Make sure to take it out of there. Let Chapter 25 focus on the analysis of low-order polynomials by differentiation and the interpretation of terms.

We have focused in this book on a small set of basic modeling functions and three operations for assembling new functions out of old ones: ***linear combination***, ***multiplication***, and ***composition***. All of these have a domain that is the whole number line, or the positive half of the number line, or perhaps the whole number line leaving out zero or some other isolated point. Consider such domains to be ***global***.

We also discussed the components of ***piecewise functions***. Each component is a function defined on a limited domain, an interval $a \leq x \leq b$. In contrast to the global domains, we'll call the limited domains ***local***.

In this chapter, we'll explore a simple and surprisingly powerful method to approximate any function ***locally***, that is, over a small domain.

Why did you?

Why would you want to approximate a function? Why not just use the function itself?

It's often the case that we know about or hypothesize about relationships only from data. We believe there is a definite functional form for the relationship, but it's unknown and unknowable to us. Still, we can approximate even an unknown

function, matching the approximation to the data that is the visible manifestation of the unknown function. Local approximations provide a general-purpose method for creating functions that can represent a wide range of relationship patterns, even ones that are not otherwise known to us.

In fields such as physics or engineering, there are often theories that dictate a particular form of function. For example, Newton's universal law of gravitation posits an inverse square law for the force of gravity as a function of distance. Mechanical engineers use power laws to describe the shape of a beam under load, and communications engineers (and others) make extensive use of sinusoids. Textbooks in those fields rightfully emphasize those particular function forms.

The utility of the local approximation method is that you can move forward even in the absence of a detailed theory. You need only apply your insight to posit which quantities are related to each other and then apply the approximation methods to produce a functional form. This approach is ubiquitous in all fields.

Sometimes, the local approximation *becomes* the theory. This is seen, for instance, in Newton's law of cooling, in Hooke's law relating force and extension, or the chemist's law of mass action.

The information that you have about the relationship often takes the form of a data table. Each row records one trial in which the values of the inputs have been measured and the corresponding output value recorded. We'll discuss the methods of constructing functions to match such data in Block 5 of this course.

Another common form for the information about the relationship is about derivatives. That is, you know something about the derivative of a relationship even though you don't (yet) have a form for the function describing the relationship. As an example, think about building a model of the sustainable speed of a bicycle as a function of the gear selected and the grade of the road—up or down.

Consider these three questions that any experienced bicyclist can likely answer:

1. On a given grade of road, is there an optimal gear for the highest sustained speed? (Have in mind a particular rider, perhaps yourself.)
2. Imagine that the grade of the road is described by a positive number for uphill and a negative number for downhill: that is, the slope of the road. For a positive (uphill) grade and at a fixed gear, will the bike's sustained speed be higher or lower as a function of the grade?[1]
3. Assuming you answered "yes" to question (1): Does the optimal gear choice depend on the grade of the road? (In concrete terms, would you choose different gears for an uphill climb than for a level road or a downhill stretch?)

Using the methods in this chapter, the answers to those three questions let you choose an appropriate form for the speed(gear, grade) function. Then, using methods in Block 5 of this text, you can make a few measurements for any given rider and construct a model customized to that rider.

Note that the three questions all have to do with derivatives. An "optimal gear" is a gear at which $\partial_{\text{gear}}\text{speed}(\text{gear}, \text{grade}) = 0$. That you ride slower the higher the numerical value of the slope means that $\partial_{\text{grade}}\text{speed}(\text{gear}, \text{grade}) < 0$. And we know that $\partial_{\text{gear}}\text{speed}(\text{gear}, \text{grade})$ depends on the grade; that's why there's a different optimal gear at each grade.

12.3 Eight simple shapes

In many modeling situations with a single input, you can get very close to a good modeling function $f(x)$ by selecting one of ***eight simple shapes***, shown in Figure @ref(fig:eight-simple-shapes).

[1] It's much the same for downhill biking, but you have to keep in mind that a shallow downhill has a higher numerical slope than a steep downhill. That is, the derivative of the hill is near zero for a very shallow grade and far from zero (that is, more negative) for a steep downhill grade.

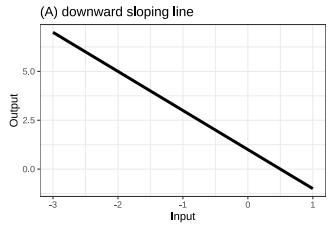


Figure 12.1: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

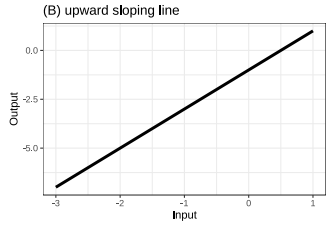


Figure 12.2: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

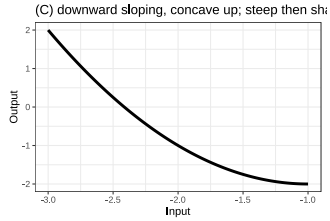


Figure 12.3: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

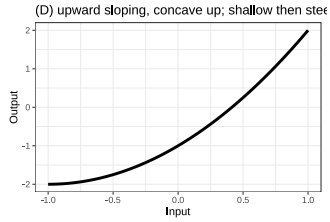


Figure 12.4: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

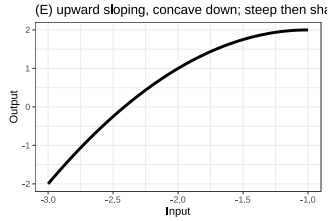


Figure 12.5: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

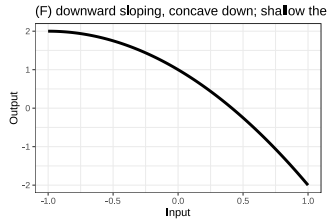


Figure 12.6: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

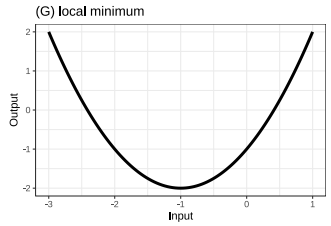


Figure 12.7: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

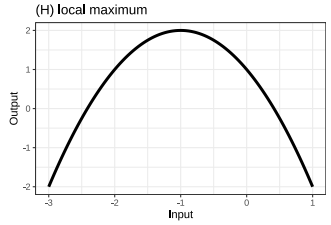


Figure 12.8: The ***eight simple shapes***, locally, of functions with one input. (See Chapter @ref(local-approximations).)

To choose among these shapes, consider your modeling context:

- is the relationship positive (slopes up) or negative (slopes down)
- is the relationship monotonic or not
- is the relationship concave up, concave down, or neither

Some examples, scenarios where the modeler knows about the derivative and concavity of the relationship being modeled. It's often the case that your knowledge of the system comes in this form.

- The incidence of an out-of-control epidemic versus time is concave up, but shallow-then-steep. As the epidemic is brought under control, the decline is steep-then-shallow and concave up. Over the whole course of an epidemic, there is a maximum incidence. Experience shows that epidemics can have a phase where incidence reaches a local minimum: a decline as people practice social distancing followed by an increase as people become complacent.
- How many minutes can you run as a function of speed? Concave down and shallow-then-steep; you wear out faster if you run at high speed. How far can you walk as a function of time? Steep-then-shallow and concave down; your pace slows as you get tired.
- How does the stew taste as a function of saltiness. The taste improves as the amount of salt increases … up to a point. Too much salt and the stew is unpalatable.
- The temperature of cooling water or the emission of radioactivity as functions of time are concave up and steep-then-shallow.
- How much fuel is consumed by an aircraft as a function of distance? For long flights the function is concave up and shallow-then-steep; fuel use increases with distance, but the amount of fuel you have to carry also increases with distance and heavy aircraft use more fuel per mile.

- In micro-economic theory there are ***production functions*** that describe how much of a good is produced at any given price, and ***demand functions*** that describe how much of the good will be purchased as a function of price.

  - As a rule, production increases with price and demand decreases with price. In the short term, production functions tend to be concave down, since it's hard to squeeze increased production out of existing facilities.

  - For demand in the short term, functions will be concave up when there is some group of consumers who have no other choice than to buy the product. An example is the consumption of gasoline versus price: it's hard in the short term to find another way to get to work. In the long term, consumption functions can be concave down as consumers find alternatives to the high-priced good. For example, high prices for gasoline may, in the long term, prompt a switch to more efficient cars, hybrids, or electric vehicles. This will push demand down steeply.

12.4 Low-order polynomials

There is a simple, familiar functional form that, by selecting parameters appropriately, can take on each of the eight simple shapes: the ***second-order polynomial***.

$$g(x) \equiv a + bx + cx^2$$

As you know, the graph of $g(x)$ is a parabola.

- The parabola opens upward if $0 < c$. That's the shape of a ***local minimum***.
- The parabola opens downward if $c < 0$. That's the shape of a ***local maximum***

Consider what happens if $c = 0$. The function becomes simply $a + bx$, the straight-line function.

- When $0 < b$ the line slopes upward.
- When $b < 0$ the line slopes downward.

With the appropriate choice of parameters, the form $a+bx+cx^2$ is capable of representing four of the ***eight simple shapes***. What about the remaining four? This is where the idea of ***local*** becomes important. Those remaining four shapes are the sides of parabolas, as in Figure @ref(fig:four-shapes).

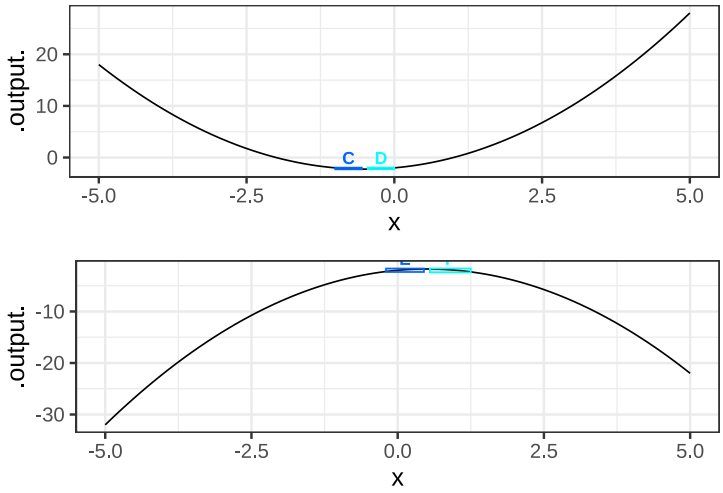


Figure 12.9: Four of the ***eight simple shapes*** correspond to the sides of the parabola. The labels refer to the graphs in Figure @ref(fig:eight-simple-shapes).

12.5 The low-order polynomial with two inputs

For functions with two inputs, the low-order polynomial approximation looks like this:

$$g(x, y) \equiv a_0 + a_x x + a_y y + a_{xy} xy + a_{yy} y^2 + a_{xx} x^2$$

In reading this form, note the system being used to name the polynomial's coefficients. First, we've used a as the root name of all the coefficients. Sometimes we might want to compare

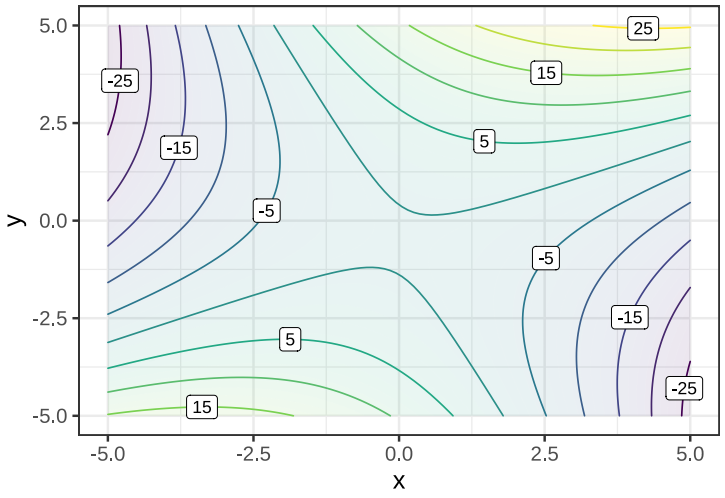
two or more low-order polynomials, so it's convenient to be able to use a for one, b for another, and so on.

The subscripts on the coefficients describes exactly which term in the polynomial involves each coefficient. For instance, the a_{yy} coefficient applies to the y^2 term, while a_x applies to the x term.

Each of a_0, a_x, a_y, a_{xy}, a_{yy}, and a_{xx} will, in the final model, be a constant quantity. Don't be confused by the use of x or y in the name of the coefficients. Each coefficient is a constant and not a function of the inputs. Often, your prior knowledge of the system being modeled will tell you something about one or more of the coefficients, for example, whether it is positive or negative. Finding a precise value is often based on quantitative data about the system.

It helps to have different names for the various terms. It's not too bad to say something like, "the a_{xy} term." (Pronounciation: "a sub x y" or "a x y") But the proper names are: ***linear terms***, ***quadratic terms***, and ***interaction term***. And a shout out to a_0, the ***constant term***.

$$g(x,y) \equiv a_0 + \underbrace{a_x x + a_y y}_{\text{linear terms}} + \underbrace{a_{xy} xy}_{\text{interaction term}} + \underbrace{a_{yy} y^2 + a_{xx} x^2}_{\text{quadratic terms}}$$



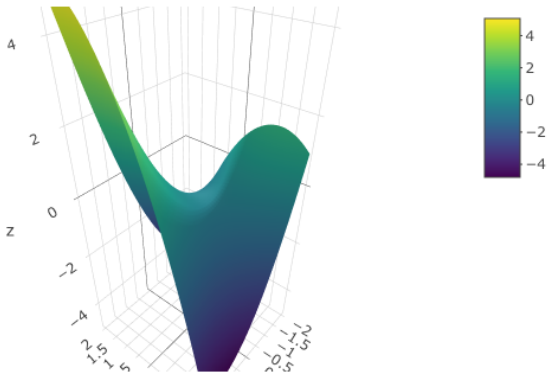


Figure 12.10: A saddle

Math in the world ...

If you're like many people, you find it harder to walk uphill than down, and find it takes more out of you to walk longer distances than shorter. Let's build a model of this, using nothing more than your intuition and the method of low-order polynomial approximations.

Let's call the map distance walked d. ("Map distance" is the horizontal change in position, disregarding vertical changes.) The steepness of the hill will be the "grade" g, which is measured as the horizontal distance covered divided by the vertical climb. If you're going downhill, the grade is negative.

The key ingredient in the model: We'll measure the "difficulty" or "exertion" to walking as the ***energy consumed*** during the walk: $E(d, g)$.

Some assumptions about walking and energy consumed:

1. If you don't walk, you consume zero energy walking.
2. The energy consumed should be proportional to the length of the walk. This is an assumption, and is probably valid, only for walks of short to medium distances, as opposed to forced marches over tens of miles.

We'll start with the full 2nd-order polynomial in two variables, and then seek to eliminate terms that aren't needed.

$$E_{big}(d,g) \equiv a_0 + a_d\, d + a_g\, g + a_{dg}\, d\, g + a_{dd}\, d^2 + a_{gg}\, g^2$$

According to assumption (1), when $E(d = 0, g) = 0$. Of course, if you are walking zero distance, it doesn't matter what the grade is; the energy consumed is still zero.

Consequently, we know that all terms that don't include a d should go away. This leaves us with

$$E_{medium}(d,g) \equiv a_d\, d + a_{dg}\, d\, g + a_{dd}\, d^2 = d\left[a_d + a_{dg}\, g + a_{dd}\, d\right]$$

Assumption (2) says that energy consumed is proportional to d. The multiplier on d in $E_{medium}()$ is $\left[a_d + a_{dg}\, g + a_{dd}\, d\right]$ which is itself a function of d. A proportional relationship implies a multiplier that doesn't depend on the quantity itself. This means that $a_{dd} = 0$.

This leaves us with a very simple model:

$$E(d,g) \equiv \left[a_1 + a_2\, g\right]\, d$$

where we have simplified the labeling on the coefficients since there are only two in the model.

Perhaps assumption (2) is mis-placed and that the energy consumed per unit distance in a walk increases with the length of the walk. If so, we would need to return to the question of a_{dd}. This is typical of the modeling cycle. Trying to be economical with model terms highlights the question of which terms are so small they can be ignored.

For instance ...

In selecting cadets for pilot training, two criteria are the cadet's demonstrated flying aptitude and the leadership potential of

the cadet. Let's assume that the overall merit M of a candidate is a function of flying aptitude F and leadership potential L.

Currently, the merit score is a simple function of the F and L scores:

$$M_{current}(F, L) \equiv F + L$$

The general in charge of the training program is not satisfied with the current merit function. "I'm getting too many cadets who are great leaders but poor pilots, and too many pilot hot-shots who are not good leaders. I would rather have an good pilot who is a good leader than have a great pilot who is a poor leader or a poor pilot who is a great leader." (You might reasonably agree or disagree with this point of view, but the general is in charge.)

The general has tasked you to revise the formula to better match her views about the balance betwen flying ability and leadership potential.

How should you go about constructing $M_{improved}(F, L)$?

You recognize that $F + L$ is a low-order polynomial: just the linear terms are present without a constant or interaction term or quadratic terms. Low-order polynomials are a good way to approximate any formula locally, so you have decided to follow that route.

Quadratic terms are appropriate when a model needs to feature a locally ***optimal*** level of the of the inputs. But it will never be the case that a lower flying score will be more favored than a higher score, and the same thing for the leadership score. So your model doesn't need quadratic terms.

That leaves the interaction term as the way forward. The low-order polynomial model will be

$$M_{improved}(F, L) \equiv d_0 + F + L + d_{FL}FL$$

Should d_{FL} be positive or negative?

Imagine a cadet Drew with acceptable and equal F and L scores. Another cadet, Blake, has scores that are $F+\epsilon$ and $L-\epsilon$, where ϵ might be positive or negative. Under the original formula for merit, Drew and Blake have equal merit. Under the new

criteria, Drew should have a higher merit than Blake. In other words:

$$M_{improved}(F, L) - M_{improved}(F + \epsilon, L - \epsilon) > 0$$

Replace $M_{improved}(F, L)$ with the low-order polynomial approximation given earlier.

$$\underbrace{d_0 + F + L + d_{FL} FL}_{M_{improved}(F,L)} - \underbrace{\left[d_0 + [F + \epsilon] + [L - \epsilon] + d_{FL}(FL - \epsilon L + \epsilon F - \epsilon^2)\right]}_{M_{improved}(F+\epsilon, L-\epsilon)} > 0$$

Collecting and cancelling terms in the above gives

$$-d_{FL}(\epsilon(F - L) + \epsilon^2) > 0$$

Since F and L were assumed equal, this results in

$$M_{improved}(F, L) - M_{improved}(F + \epsilon, L - \epsilon) = d_{FL}\, \epsilon^2 > 0$$

Thus, d_{FL} will have to be positive.

12.6 Two-variable modeling polynomial

In Section @ref(modeling-polynomial-1) we introduced the low-order polynomial, either $g_1(x) \equiv a_0 + a_1 x$ or $g_1(x) \equiv b_0 + b_1 x + b_2 x^2$ as a general-purpose way of generating a function with a smoothly curved shape. The same applies in constructing simple functions of two variables.

Almost always, you should use at least a first-order polynomial, which is:

$$h_1(x, y) \equiv a_0 + a_x x + a_y y$$

But there is an important extension of this, using what's called a ***bilinear term*** or, more evocatively in statistics, an ***interaction term***. This is

$$h_2(x, y) \equiv \underbrace{b_0}_{\text{intercept}} + \underbrace{b_x\, x + b_y\, y}_{\text{linear terms}} + \underbrace{b_{xy}\, x\, y}_{\text{bilinear term}}$$

The bilinear term arises in models of phenemona such as the spread of epidemics, the population dynamics of predator and prey animals, and the rates of chemical reactions. In each of these situations one thing is interacting with another: a predator killing a prey animal, an infective individual meeting a person susceptible to the disease, one chemical compound reacting with another.

Under certain circumstances, modelers include one or both ***quadratic terms***, as in

$$h_3(x,y) \equiv c_0 + c_x\, x + c_y\, y + c_{xy}\, x\, y + \underbrace{c_{yy}\, y^2}_{\text{quadratic in y}}$$

The skilled modeler can often deduce which terms to include from basic facts about the system being modeled. We'll need some additional calculus concepts before we can explain this in a straightforward way.

WHY DID YOU?

In writing polynomials like

$$h_1(x,y) \equiv a_0 + a_x x + a_y y$$

or

$$h_3(x,y) \equiv c_0 + c_x x + c_y\ y + c_{xy} x\ y$$

we are using **letters** as subscripts on the coefficients. Think of c_x as saying, "I am the coefficient on the x term in the polynomial." Using this style lets us use different letters from the start of the alphabet for the names of coefficients in the different polynomials while still making it clear which term each coefficient is scaling.

12.7 What's wrong with polynomials

12.8 Exercises

13 Operations

Graphs are relatively modern, coming into mainstream use only in the 1700s. Much of mathematics was developed before graphs were invented. One consequence of this is that function tasks that are easy using a graph might be very hard with the previous ways of implementing functions. This is analogous to the way that arithmetic is pretty easy with Arabic numerals, but really hard with Roman numerals.

WHEN INTRODUCING Zeros() or argM(), we need to name the output so that we can use it in another part of the model. MODIFY THIS: We will also use subscripts and non-numeric superscripts with input names to identify specific, isolated input quantities that are significant in some way. So you'll see x_0 or $y^\star$. For instance, such notation will be used for an input value where the function output is zero.

13.1 Outline

13.1.1 Function as input

These are operations that take a ***function as input***. Depending on the operation, the object returned might be a scalar, a data frame, or another function. We use a data frame when there might be more than one quantity produced as output.

Argmax/min — returns a data frame Solve/Invert/Zeros – returns a data frame Slopefun — returns a function D/antiD — returns a function Integrate — returns a quantity

13.1.2 Data frame as input

Returns a function: spliner, fitModel

13.2 Data and functions

In Figure @ref(fig:penguin-flipper), the data point plot of the penguin flipper length vs body mass, there are generally multiple penguins with the same body mass but different flipper lengths. The overall impression is that of a ***cloud of points***.

When we construct a function to model the pattern observed in that cloud, we need to respect the mathematical definition of function, part of which is that a function has ***only one output*** for any given input.

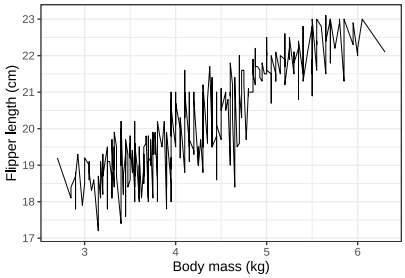


Figure 13.1: Connecting the penguin data points with lines (left) doesn't directly describe any sensible mathematical function. We use modeling to create a smooth function (right) that stays close, but not too close, to the data points.

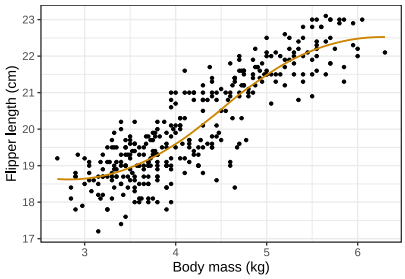


Figure 13.2: Connecting the penguin data points with lines (left) doesn't directly describe any sensible mathematical function. We use modeling to create a smooth function (right) that stays close, but not too close, to the data points.

To create the model of flipper length as a function of body mass, we used one of a set of techniques called ***machine learning***. That is, we didn't specify that the form should be an exponential or a gaussian or a sigmoid or any other particular shape. We simply asked the computer to figure out a smooth function that stays close to the data. The result, as it happens, was a sigmoid.

13.3 Inverting a function

A function graph makes it easy to evaluate the ***function inverse***. For all the basic modeling functions we have a way to calculate numerically the output for any given input (in the function's domain).

You can easily evaluate a function for a given input from its graph. As you know, just put your finger at the horizontal coordinate for the input. Then move your finger upward to reach the point on the curve directly above that horizontal coordinate. You read off the value of the function at that input by reference to the scale on the vertical axis.

Often, working with a function goes another way: you know the output and you want to find a corresponding input. It's easy to do this with a graph. Pick the position on the vertical axis that represents the given input. Then trace horizontally to where the ink is. From there, trace vertically to read off the value of an input that would produce the given output.

Mathematicians are careful to distinguish between functions where there is one and only one possible input that generates each given output, and functions where there can be more than one input that generates the same output. (Mathematicians use the word ***unique*** to mean "one and only one.") Functions with a one-to-one relationship between output and input are called "invertible."

Invertible or not, it is a common procedure for working with functions to find *an* input corresponding to a specific, given output. In high-school algebra, this was called "solving for x." A special case of solving is finding the ***roots of a polynomial***. The name we give to the procedure is ***zero finding***, which correctly points out that we are trying to **find** an input.

Very much in the spirit of naming common tasks, the process of turning an output from a function into the corresponding input is called ***solving***. In high-school mathematics, you may have heard the phrases "solve for x" or "find x." Generally what was mean can be expressed using functions this way: Given an output $g(x_0)$ find the input x_0 that would produce that output.

In computer programming, "solving" is sometimes called ***zero finding***. As is often the case, computer programming often involves re-arranging things to fit in a standard format. Here, the standard format is to find the input that corresponds to an output of *zero*. In other words, given only the output $A = g(x_0)$ and not x_0 itself, find a value for x_0 that satisfies look for the input x_0 at which $g(x_0) - A = 0$.

13.4 Function inverses

This may sound like a familiar word problem from your high-school algebra course:

> You are a facilities manager for a small town. The town contains approximately 400 miles of road that must be plowed following a significant snowfall. How many plows must be used in order to complete the job in one day if the plows can travel at approximately 7 miles per hour when engaged? — Source

The task of answering such a question is often called ***solving*** a ***word problem***.

You don't need calculus to solve this problem, but insofar as the format is familiar to you, it might help to depict how it would

be addressed as a modeling task and how to use the model created to guide the mathematical work of getting a numerical answer to the problem.

Modeling Phase: The objective is to create a function that represents snow plowing and that will let us answer the question about how much plowing activity is needed. Here's a function that takes as input x a number of plows and provides as output the number of miles that can be serviced in a day.

$$\text{miles.plowed}(x) \equiv 7 \text{ mph} \times 24 \text{ hours-per-day} \times x \text{ plows}$$

In reality, "miles plowed" depends on the amount of snow, the safe speed limit of the plows, the number of rest breaks needed by the drivers, how far the snowplow terminus is from the road system, how many cars are parked on the road and the available number of tow trucks, and the day of the week and the time of day. (Remember, there might be other traffic on the road. Plowing at rush hour is bound to be slow! And the reduced visibility at night calls for extra care!) It seems that textbook "word problems" never mention such issues, having been written in a world where plowing snow is exactly the same as doing simple arithmetic.

Likely, we're going to have to use the ***modeling cycle*** to end up with a genuinely useful model. Still, we have to start somewhere, so let's start with $7 \times 24 \times x$.

We can use this function to solve the problem: How many plows are needed to get the 400-miles of road serviced in 1 day? It's a matter of choosing a suitable method for applying the function to guide us to the answer.

One simple method, which sometimes is called ***guess and check*** is to propose some answers and see what happens. Being experts in snow plowing, we know that you can't have negative or fractional plows, so our guesses for x will be integers. Let's do this systematically:

Number of plows x	Miles plowed in a day
1	$7 \times 24 \times 1 = 168$
2	$7 \times 24 \times 2 = 336$
3	$7 \times 24 \times 3 = 504$

Number of plows x	Miles plowed in a day
4	$7 \times 24 \times 4 = 672$
$\vdots$	… and so on.

We really don't need all the scratch work crowding up the table, so let's streamline it, keeping the essentials:

input x	output miles.plowed(x)
1	168
2	336
3	504
4	672

The domain of the miles_plowed() function is $x = 0, 1, 2, 3, ...$, so our table covers only a bit of the domain.

Our purpose in constructing miles_plowed() is not to figure out the output given the input, but to do the opposite: given the output, find the input. The information we have is in the form of the output: 400 miles. The mathematical operation of ***solving*** consists of looking up what we do know in the **output** column of the table, then reading off the corresponding **input** as our answer. Since 400 doesn't appear in the output column, we'll look for an interval that includes 400. Of course, that's the interval from 336 to 504. So the answer will be something bigger than $x = 2$ but doesn't need to be any larger than $x = 3$. You've been in the facilities management business for many years, so you know to choose the answer $x = 3$.

If you have a graph of a function, it can be easy to calculate what the input should be for a given output. Just reverse the finger action, looking up the output on the vertical axis, tracing horizontally to the function graph, then reading off the result from the input axis.

Now let's write the problem using math notation rather than a table. We're looking for a value of x such that

$$\text{miles.plowed}(x) = 400$$

This is an ***equation*** as opposed to a ***function definition***. We write equations with = and definitions with ≡. The point of writing equations is often to signal to us that the task is to "solve for x."

An algebraic solution relies on replacing miles_plowed(x) with the function's formula and then re-arranging numbers and possibly other symbols until we have an equation of the form $x = ...$ Here, that's easy:

$$7 \times 24 \times x = 400 \quad \Longrightarrow \quad x = 400/(7 \times 24)$$

Apply some arithmetic and we find $x = 2.381$. Apply some common sense and we translate this into "three plows."

An industrious facilities manager might go further. "Today there is 400 miles of road. But next year there will be more, although I don't yet know the exact number. While I'm doing all this math work, I'll write up a memo with a formula so that next year, when I know how much road there will be, my assistant can find the answer with simple arithmetic."

You likely learned how to set up and solve the manager wants to put in her memo. The key is to replace 400 with a symbol standing for the number of miles of road. We'll use M. Then you do the re-arrangement with M in place of 400.

$$7 \times 24\, x = M \quad \Longrightarrow \quad x = M/168$$

All that remains is to give a more informative name in place of x and to write it as a proper function: $n_{plows}(M) \equiv M/168$.

This process of starting with a function like miles.plowed(x) and transforming it into a function in a more convenient format for the task at hand $n_{plows}(M)$ is called ***inverting the function***.

High-school algebra emphasizes techniques for inverting functions by moving symbols around. This is great when it can be done, but it's often impossible in real science and engineering problems. The ***guess and check*** method that was used in the snowplow mileage table is very simple. particularly when it's

paired with a process for refining the guess after every check. ***Newton's method*** is one such method, it makes use of the idea of slope functions introduced in Block 2.

Another common strategy for inverting functions that are beyond our own reach algebraically is to rely on a specialist to develop the inverse function, give it a name, write down a formula in mathematical notation or, sometimes more usefully, write a computer program that implements the inverse function's algorithm. With this strategy, all that's needed to invert a function is to know the name or formula of the function's inverse. Here's a table of such names and formulas, most of which is likely familiar to you.

function	inverse function
Exponential: e^x	Logarithm ("natural"): $\ln(x)$
Exponential: 2^x	Log base 2: $\log_2(x)$
Exponential: 10^x	Log base 10: $\log_{10}(x)$.
Power law: x^2	Square root/power-law: $\sqrt{x} = x^{1/2}$
Power law: x^{-1}	Power law: x^{-1}
Trig: $\sin(x)$	Trig: $\arcsin(x)$
Trig: $\tan(x)$	Trig: $\arctan(x)$

You can of course read this table either from left to right or from right to left. For instance, the inverse of $\ln(x)$ is e^x.

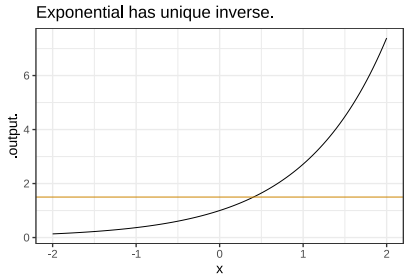
As you know, for a function $g(x)$, the set of valid values of x is called the ***domain*** of the function. The set of possible outputs from the function is called the ***range*** of the function. The inverse function to $g()$ has a domain which is the range of $g()$ and a range which is the domain of $g()$.

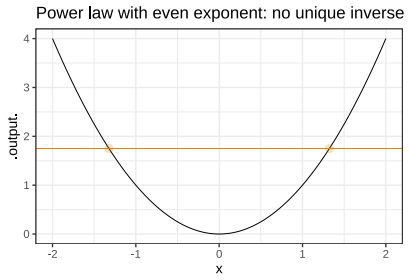
Sometimes the notation $g^{-1}(x)$ is used for "the inverse function of $g()$. This is potentially confusing, since $g^{-1}(x)$ might reasonable be interpretted as $1/g(x)$ and, in general, those two things are not equal.

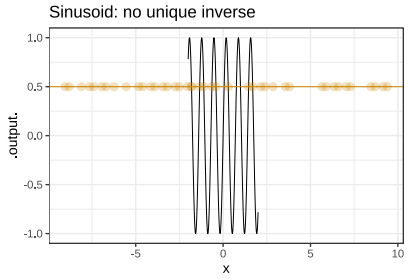
Straight-line (with non-zero slope), exponential (e^{kt} where $k \neq 0$), and logarithm functions aways have inverse functions. For power-law and sinusoid functions, there can be several (or

many) different inverse functions. We'll get to this when we need to.

There's an easy graphical test for whether there is a unique inverse function or not. Draw the graph, then see if any horizontal line touches the graph of the function in more than one place. If so, there is no unique inverse function.







13.5 Solving graphically

Most readers are familiar with the sort of high-school math problem where you are presented with an equation written in

terms of x and asked to "find x" or "find the solution." For instance, the problem might be

$$\text{Let } \; 3x - 2 = 7x^2. \quad \text{Find } x.$$

The above problem falls into one class of such tasks, the class of "finding roots of polynomials." Here's another problem in that same class:

> *An investment club decided to buy $9000 worth of stock with each member paying an equal share. But two members left the club, and the remaining members had to pay $50 more apiece. How many members are in the club?*[1]

Let's pose the problem in terms of a function, money_raised(x) where x is the original number of members in the investment club. Originally, the members were each going to pay $9000/x$. But after the two people drop out, each of the remaining members has to pay $9000/x + 50$. Since there are $x - 2$ remaining members, the total amount raised is

$$\text{money.raised}(x) \equiv (x-2)\left(\frac{9000}{x} + 50\right)$$

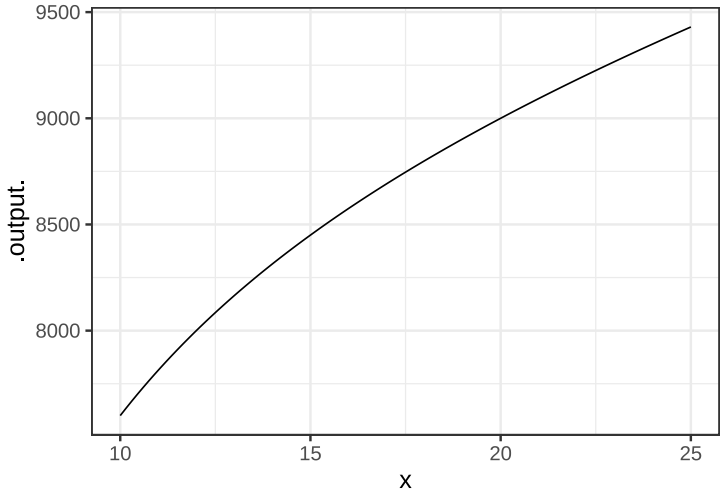
Figure @ref(fig:money-raised) graphs money.raised(x) ::: {.cell layout-align="center" fig.showtext='true'}

```
money_raised <- makeFun((x-2)*(9000/x + 50) ~ x)
slice_plot(money_raised(x) ~ x, domain(x=c(10,25)))
```

[1] This translates to an equation, where x is the number of members:

$$(x-2)(9000/x + 50) = 9000$$

Source



::: The function tells us, for any number of members x, how much money will be raised. So what's x for the club? The function doesn't tell us … until we add in a new fact. The club raised \$9000. So look up \$9000 on the vertical axis, trace horizontally over to the graph, and read off the resulting input that generates an output of \$9000.

This inverse way of using functions—we know the output and we need to calculate the input—is very common. One reason is that sometimes it's relatively easy to construct a model that takes input x to an output, but the problem at hand is to figure out from a value of the output what is the corresponding value of the input.

Math in the world …

Seismograms and electrocardiograms

Suppose that you have a function $f(x)$ and you also know the inverse of that function. We'll call the inverse $g(x)$ for this example, but a more typical notation is $f^{-1}(x)$ where the function name is f^{-1}.

Composing a function and its inverse gives back the input: $f(g(x)) = x$ and $g(f(x)) = x$.

As part of your high-school mathematics education, you learned the inverses to several functions. For instance, e^x and $\ln(x)$ are inverses, meaning that $\ln(e^x) = x$ and $e^{\ln(x)} = x$. The identity function is its own inverse. The inverse of the square function is the square root: $\sqrt{x^2} = x$ and $(\sqrt{x})^2 = x$.

It's helpful to keep in mind that if you have a function $f(x)$ there is some other function—although you may not know its form yet—that is the inverse to $f()$.

There are two caveats:

i. Mathematicians will say that for functions like $\sin(x)$ or $\text{dnorm}(x)$ which fail the horizontal line test the inverse does not exist. That's because mathematicians want the inverse to have a single, unique value, while functions like dnorm() have two inputs that generate any output in the function's range. sin() has an infinite number of inputs that generate any output in the function's range.

   But in practice, the problem is to find **some** input that will generate the known output. If there is more than one such input, we just have to choose one.

   We can even satisfy the mathematician's insistence that there be a unique inverse if we restrict the domain of the forward function to a region where it is monotonically increasing or monotonically decreasing.

ii. There is always an algorithm to that can be applied to find a solution to any inverse problem. But this algorithm is rarely a simple formula. Not having a formula, we play a name game. For instance, the inverse of the square function is called the "square root," but there is no simple formula for finding the square root of a number. Similarly, the inverse to $\sin(x)$ is called $\arcsin(x)$, but there is no simple formula for arcsin(). Almost invariably we use such inverse functions not by doing arithmetic to calculate their output, but by using a calculator button or a software function that carries out the algorithm.

For now, it suffices that you can compute the inverse to any function (over some sub-domain where the function is monotonically increasing or decreasing) by graphing the function and then reading from the vertical axis to the horizontal. As we encounter more concepts from calculus and computing, we'll develop additional methods for evaluating an inverse function without first having to construct that function.

13.6 Zero-finding

Zero-finding is the name of a process for finding a function input that produces a function output of zero. Usually, zero isn't a an output value of interest. (If the snow-plow manager asked her assistant to figure out how many plows are needed to cover 0 miles in one day, the assistant would wonder if someone has lost their mind.)

However, any solving problem can be translated into a zero-finding problem. For example, if the problem is to solve $g(x) = M$, we can equally well state the problem as finding a zero of $h(x) \equiv g(x) - M$.

WHY DID YOU?

Why would anyone want to do this?

Often, mathematical algorithms in computer languages are written to accept a ***function*** as an input rather than an ***equation***. For instance the R/mosaic operator `findZeros()` takes as it's primary argument a tilde expression describing a function.

For example, the investment-club problem given earlier in this chapter can be re-stated as a zero-finding problem.

```
money_raised <- makeFun((x-2)*(9000/x + 50) ~ x)
Zeros(money_raised(x) - 9000 ~ x, domain(x=-100:100))
```

```
## # A tibble: 3 x 2
##              x .output.
##          <dbl>    <dbl>
## 1 -18.0        1.88e-7
## 2   0.0000665 -2.71e+8
## 3  20.0        4.09e-8
```

It turns out that there are two zeros for 'money_raised(x) - 9000$. Only one of them has any sensible interpretation as the number of investors in the club.

13.7 Exercises

```
rr insert_calcZ_exercise("6.1", "9LB8hK", "Exercises/beech-lead-piano.Rmd")

rr insert_calcZ_exercise(10.1, "IWLDN1", "Exercises/inverse-flip-1.Rmd")

rr insert_calcZ_exercise("10.3", "1kZXxT", "Exercises/titmouse-throw-sofa.Rmd")

rr insert_calcZ_exercise("XX.XX", "9s6ntY", "snail-stand-plate.Rmd",
skill = "functions have a unique output for each
input")
```

14 Magnitude

Undoubtedly you are comfortable with the standard way of writing numbers, for instance 33 or 512 or 1051. Elementary school students master the comparison of such numbers. Which is greater: 512 or 33? Which is less, 1051 or 512? You can answer such questions at a glance; the comparison can be accomplished simply by counting the number of digits. 1051 has four digits, so it is larger than the three-digit number 512. There are two digits in 33, so it smaller than 512. When two numerals have the same number of digits—say, 337 and 512—you can't answer the "greater than" question by simple counting. Instead, you proceed from left to right and compare the number in each place. So, for 512 and 337, you compare 5 to 3 and … since 5 is greater than 3, 512 is greater than 337. If the two leading digits are the same, go on to the next digit and so on for all the digits in turn.

Things were not always this simple. Our number system that uses *place* and *Arabic* numerals is a human invention. An example of an earlier number system is Roman numerals. Here, comparison is hard. For instance, which of these three numbers is bigger?

MLI or CXII or XXXIII

The typographically shorter number is the largest, and vice versa. Even when two Roman numerals have the same length, it's not trivial to compare them on a place-by-place basis. For instance, IC is about fifteen times bigger than VI, even though I is much smaller than V.

14.1 Counting digits

Digit counting provides an easy, fast way to perform many calculations, at least approximately. What is $\sqrt{10000}$? There are five digits, and the square root of a number will have "half the number of digits." So, $\sqrt{10000} = 100$. What is 10×34? Easy: 340. Just append the one zero from 10 to the end of 34. What is 1000×13? Just as easy: 13,000. We even punctuate written numbers with commas and a period in order to facilitate counting digits.

Imagine having a digit counting function called digit(). It takes a number as input and produces a number as output. We don't have a ***formula*** for digit(), but for some inputs the output can be calculated just be counting. For example:

- digit(10) $\equiv$ 1
- digit(100) $\equiv$ 2
- digit(1000) $\equiv$ 3
- … and so on …
- digit(1,000,000) $\equiv$ 6
- … and on.

The digit() function easily can be applied to the product of two numbers. For instance:

- digit(1000 $\times$ 100) = digit(1000) + digit(100) = 3 + 2 = 5.

Similarly, applying digit() to a ratio gives the difference of the digits of the numerator and denominator, like this:

- digit(1,000,000 $\div$ 100) = digit(1,000,000) - digit(100) = 6 - 2 = 4

14.2 Using digit() to understand magnitude

We haven't shown you the digit() function for anything but the handful of discrete inputs listed above. It was a heroic task to produce the continuous version of digit(). The method is sketched out in @ref(fractional-digits).

In practice, digit() is so useful that it could well have been one of our basic modeling functions:

$$\text{digit(x)} = 2.302585 \ln(x)$$

or, in R, `log10()`. We elected ln() rather than digit() for reasons that will be seen when we study ***differentiation***.

14.3 Quantity and magnitude

The familiar quantity 60 miles-per-hour is written as a number (60 here) followed by units. The quantity is neither the number nor the units: it is the combination of the two. For instance, 100 is obviously not the same as 60. And miles-per-hour is not the same as kilometers-per-hour. Yet, 60 miles-per-hour is almost exactly the same quantity as 100 kilometers-per hour.[1]

6, 60, 600, and 6000 miles-per-hour are quantities that differ in size by ***orders of magnitude***. Such differences often point to a substantial change in context. A jog is 6 mph, a car on a highway goes 60 mph, a cruising commercial jet goes 600 mph, a rocket passes through 6000 mph on its way to orbital velocity.

In everyday speech, the difference between 60 and 6 is 54; just subtract. Modelers and scientists routinely mean something else: the difference between 60 and 6 is "one order of magnitude." Similarly, 60 and 6000 are different by "two orders of magnitude," and 6 and 6000 by three orders of magnitude.

In everyday English, we have phrases like "a completely different situation" or "different in kind" or "qualitatively different" (note the **l**) to indicate substantial differences. "Different orders of magnitude" expresses the same kind of idea but with specific reference to quantity.

The use of factors of 10 in counting orders of magnitude is arbitrary. A person walking and a person jogging are on the edge of being qualitatively different, although their speeds differ by a factor of only 2. Aircraft that cruise at 600 mph and 1200

[1] 95.69 km/hr is exactly 60 mph.

mph are qualitatively different in design, although the speeds are only a factor of 2 apart. A professional basketball player (height 2 meters or more) is qualitatively different from a third grader (height about 1 meter).

Modelers develop an intuitive sense for when to think about difference in terms of a subtractive difference (e.g. 60 - 6 = 54) and when to look at orders of magnitude (e.g. 60-to-6 is one order of magnitude). This seems to be a skill based in experience and judgment, as opposed to a mechanical process.

One clue that thinking in terms of orders of magnitude is appropriate is when you are working with a set of objects whose range of sizes spans one or many factors of 2. Comparing baseball and basketball players? Probably no need for orders of magnitudes. Comparing infants, children, and adults in terms of height or weight? Orders of magnitude may be useful. Comparing bicycles? Mostly they fit within a range of 2 in terms of size, weight, and speed (but not expense!). Comparing cars, SUVs, and trucks? Differences by a factor of 2 are routine, so thinking in terms of order of magnitude is likely to be appropriate.

Another clue is whether "zero" means "nothing." Daily temperatures in the winter are often near "zero" on the Fahrenheit or Celcius scales, but that in no way means there is a complete absence of heat. Those scales are arbitrary. Another way to think about this clue is whether ***negative values*** are meaningful. If so, thinking in terms of orders of magnitude is not likely to be useful.

You may have guessed that `digits()` is handy for computing differences in terms of orders of magnitude. Here's how:

1. Make sure that the quantities are expressed in the same ***units***.
2. Calculate the difference between the `digits()` of the numerical part of the quantity.

What is the order-of-magnitude difference in velocity between a snail and a walking human. A snail slides at about 1 mm/sec,

a human walks at about 5 km per hour. Putting human speed in the same units as snail speed:

$$5\frac{km}{hr} = \left[\frac{1}{3600}\frac{hr}{sec}\right] 5\frac{km}{hr} = \left[10^6\frac{mm}{km}\right]\left[\frac{1}{3600}\frac{hr}{sec}\right] 5\frac{km}{hr} = 1390\frac{mm}{sec}$$

Calculating the difference in `digits()` between 1 and 1390: ::: {.cell layout-align="center" fig.showtext='true'}

```
log10(1390) - log10(1)
## [1] 3.143015
```

So, about 3 orders of magnitude difference in speed. To a snail, we walking humans must seem like rockets on their way to orbit! :::

Math in the world ...

Animals, including humans, go about the world in varying states of illumination, from the bright sunlight of high noon to the dim shadows of a half moon. To be able to see in such diverse conditions, the eye needs to respond to light intensity across many orders of magnitude.

The ***lux*** is the unit of illuminance in the Système international. This table[2] shows the illumination in a range of familiar outdoor settings:

Illuminance	Condition
110,000 lux	Bright sunlight
20,000 lux	Shade illuminated by entire clear blue sky, midday
1,000 lux	Typical overcast day, midday
400 lux	Sunrise or sunset on a clear day (ambient illumination)
0.25 lux	A full Moon, clear night sky
0.01 lux	A quarter Moon, clear night sky

For a creature active both night and day, they eye needs to

[2]Source: https://en.wikipedia.org/wiki/Daylight

be sensitive over 7 orders of magnitude of illumination. To accomplish this, eyes use several mechanisms: contraction or dilation of the pupil accounts for about 1 order of magnitude, photopic (color, cones) versus scotopic (black-and-white, rods, nighttime) covers about 3 orders of magnitude, adaptation over minutes (1 order), squinting (1 order).

14.4 Composing $\ln()$

The logarithm is the inverse of the exponential function. In other words,

$$\ln(e^x) = x \quad \text{and} \quad e^{\ln(x)} = x$$

Think about this in terms of the kinds of quantities that are the input and output to each function.

- Logarithm: The input is a quantity, the output is the magnitude of that quantity.
- Exponential: The input is a magnitude, the output is the quantity with that magnitude.

14.5 Magnitude graphics

In order to display a variable from data that varies over multiple orders of magnitude, it helps to plot the ***logarithm*** rather than the variable itself. Let's illustrate using the `Engine` data frame, which contains measurements of many different internal combustion engines of widely varying size. For instance, we can graph engine RPM (revolutions per second) versus enging mass, as in Figure @ref(fig:rpm-mass).

```
gf_point(RPM ~ mass, data = Engines)
```

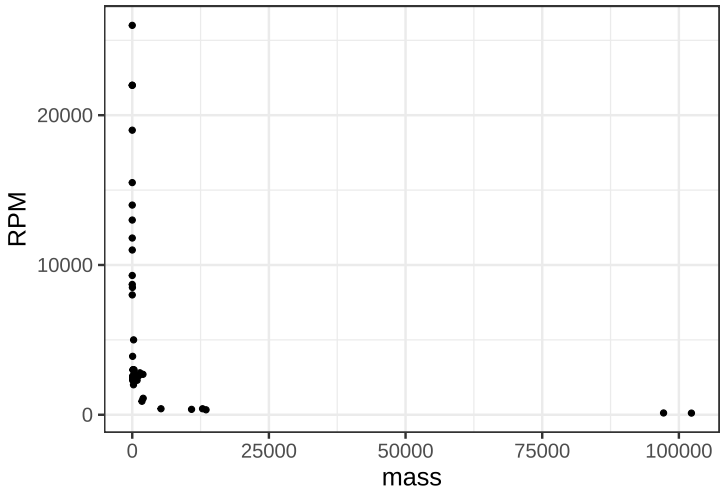


Figure 14.1: Engine RPM versus mass for 39 different enginges plotted on the standard linear axis.

In the graph, most of the engines have a mass that is … zero. At least that's what it appears to be. The horizontal scale is dominated by the two huge 100,000 pound monster engines plotted at the right end of the graph.

Plotting the logarithm of the engine mass spreads things out, as in Figure @ref(fig:rpm-mass-log).

```
gf_point(RPM ~ mass, data = Engines) %>%
  gf_refine(scale_x_log10())
```

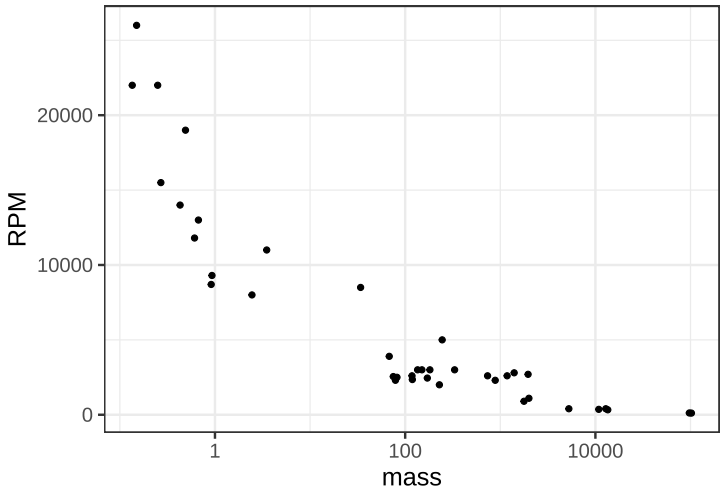


Figure 14.2: Engine RPM versus mass on semi-log axes.

Note that the horizontal axis has been labelled with the actual mass (in pounds), with the labels evenly spaced in terms of their logarithm. This presentation, with the horizontal axis constructed this way, is called a ***semi-log*** plot.

When both axes are labeled this way, we have a ***log-log*** plot, as shown in Figure @ref(fig:rpm-mass-log-log).

```
gf_point(RPM ~ mass, data = Engines) %>%
  gf_refine(
    scale_x_log10(),
    scale_y_log10()
    )
```

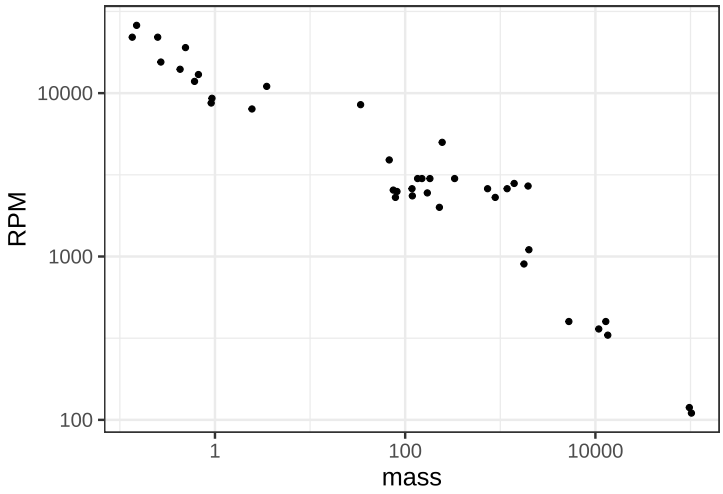


Figure 14.3: Engine RPM versus mass on log-log axes.

Semi-log and log-log axes are widely used in science and economics, whenever data spanning several orders of magnitude need to be displayed. In the case of the engine RPM and mass, the log-log axis shows that there is a graphically simple relationship between the variables. Such axes are very useful for displaying data, but can be hard for the newcomer to read quantitatively. For example, calculating the slope of the evident straight-line relationship in Figure @ref(fig:rpm-mass-log-log) is extremely difficult for a human reader and requires translating the labels into their logarithms.

Math in the world ...

Robert Boyle (1627-1691) was a founder of modern chemistry and of the scientific method in general. As any chemistry student already knows, Boyle sought to understand the properties of gasses. His results are summarized in *Boyle's Law*.

The data frame `Boyle` contains two variables from one of Boyle's experiments as reported in his lab notebook: pressure

in a bag of air and volume of the bag. The units of pressure are mmHg and the units of volume are cubic inches.[3]

Famously, Boyle's Law states that, at constant temperature, the pressure of a constant mass of gas is inversely proportional to the volume occupied by the gas. Figure @ref(fig:boyle-movie) shows a cartoon of the relationship.

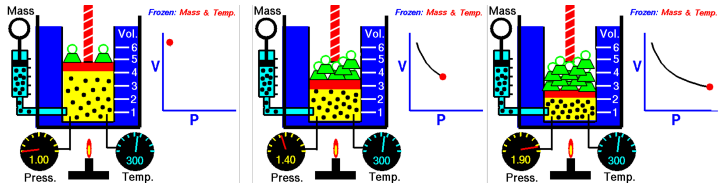


Figure 14.4: A cartoon illustrating Boyle's Law. Source: NASA Glenn Research Center

Figure @ref(fig:boyle-data) plots out Boyle's actual experimental data. I

```
gf_point(pressure ~ volume, data = Boyle) %>%
  gf_lm()
```

[3]Boyle's notebooks are preserved at the Royal Society in London. The data in the `Boyle` dataframe have been copied from this source.)

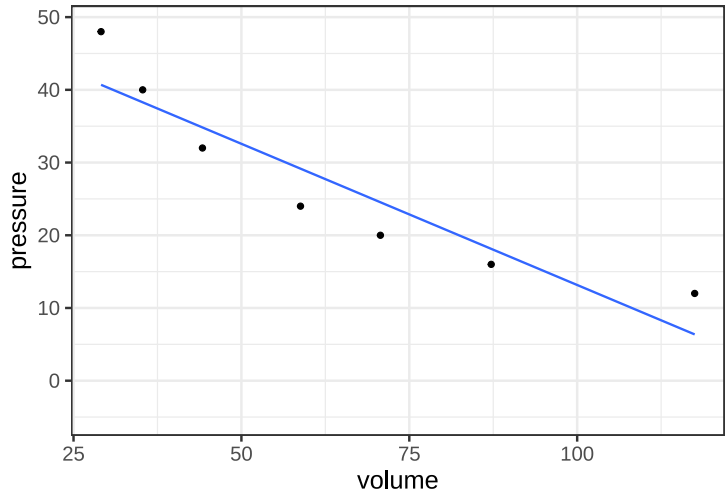


Figure 14.5: A plot of Boyle's pressure vs volume data on linear axes. The straight line model is a poor representation of the pattern seen in the data.

You can see a clear relationship between pressure and volume, but it's hardly a linear relationship.

Plotting Boyle's data on log-log axes reveals that, in terms of the logarithm of pressure and the logarithm of volume, the relationship is linear.

```
gf_point(log(pressure) ~ log(volume), data = Boyle) %>%
  gf_lm()
```

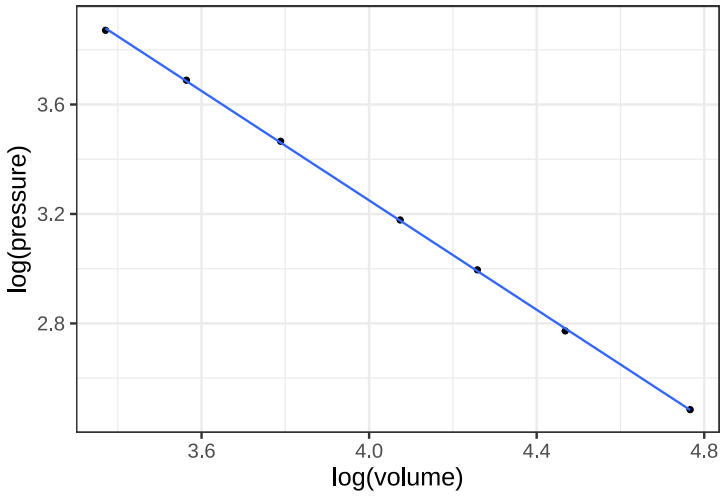


Figure 14.6: Plotting the logarithm of pressure against the logarithm of volume reveals a straight-line relationship.

Figure @ref(fig:boyle-data-log) shows that Boyle's log-pressure and log-volume data are a straight-line function. In other words:

$$\ln(\text{Pressure}) = a + b\ln(\text{Volume})$$

You can find the slope b and intercept a from the graph. For now, we want to point out the consequences of the straight-line relationship between logarithms.

Exponentiating both sides gives

$$e^{\ln(\text{Pressure})} = \text{Pressure} = e^{a+b\ln(\text{Volume})} = e^a \left[e^{\ln(\text{Volume})}\right]^b = e^a\,\text{Volume}^b$$

or, more simply (and writing the number e^a as A)

$$\text{Pressure} = A\,\text{Volume}^b$$

A power-law relationship!

14.6 Reading logarithmic scales

Plotting the logarithm of a quantity gives a visual display of the magnitude of the quantity and labels the axis as that magnitude. A useful graphical technique is to label the axis with the original quantity, letting the position on the axis show the magnitude.

To illustrate, Figure @ref(fig:mag-scales-1)(left) is a log-log graph of horsepower versus displacement for the internal combustion engines reported in the `Engines` data frame. The points are admirably evenly spaced, but it is hard to translate the scales to the physical quantity. The right panel in Figure @ref(fig:mag-scales-1) shows *exactly the same data points*, but now the scales are labeled using the original quantity.

```
gf_point(log(BHP) ~ log(displacement), data = Engines)
gf_point(BHP ~ displacement, data = Engines) %>%
  gf_refine(scale_y_log10(), scale_x_log10())
```

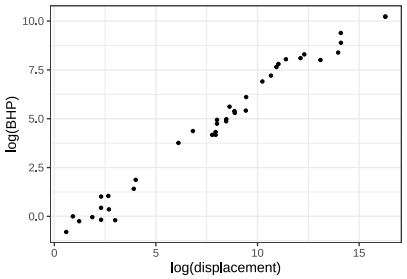


Figure 14.7: Horsepower versus displacement from the `Engines` data.frame plotted with log-log scales.

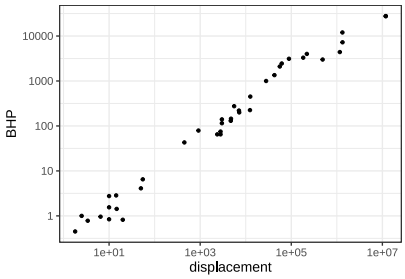


Figure 14.8: Horsepower versus displacement from the `Engines` data.frame plotted with log-log scales.

The tick marks on the vertical axis in the left pane are labeled for 0, 2.5, 5.0, 7.5, and 10. That doesn't refer to the horsepower itself, but to the logarithm of the horsepower. The right pane has tick labels that are in horsepower at positions marked 1, 10, 100, 1000, 10000.

Such even splits of a 0-100 scale are not appropriate for logarithmic scales. One reason is that 0 cannot be on a logarithmic scale in the first place since $\log(0) = -\infty$.

Another reason is that 1, 3, and 10 are pretty close to an even split of a logarithmic scale running from 1 to 10. It's something like this:

```
1              2             3          5             10     x
|-----------------------------------------------------|
0               1/3          1/2        7/10           1     log(x)
```

It's nice to have the labels show round numbers. It's also nice for them to be evenly spaced along the axis. The 1-2-3-5-10 convention is a good compromise; almost evenly separated in space yet showing simple round numbers.

14.7 Fractional digits (optional)

So far, we have the digit() function in a tabular form:

input	output
⋮	⋮
0.01	-2
0.1	-1
1	0
10	1
100	2
1000	3
10,000	4
100,000	5
1,000,000	6
⋮	⋮

Here's the point-plot presentation of the table:

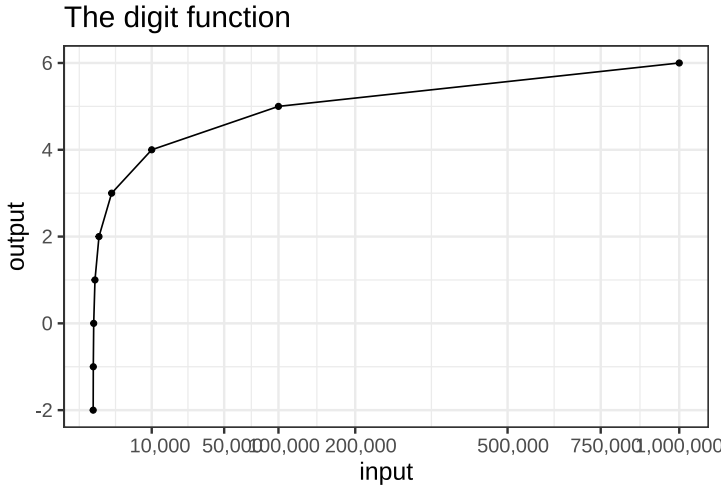


Figure 14.9: Connecting the data points for the digit function to make a continuous function.

We've imagined digits() to be a continuous function so we've connected the gaps with a straight line. Now we have a function that has an output for any input between 0.01 and 1,000,000, for instance, 500,000.

The angles between consecutive line segments give the function plotted in Figure @ref(fig:log-1st-try) an unnatural look. Still,

it is a continuous function with an output for any input even if that input is not listed in the table.

Starting around 1600, two (now famous) mathematicians, John Napier (1550-1617) and Henry Briggs (1561-1630) had an idea for filling in gaps in the table. They saw the pattern that for any of the numbers a and b in the input column of the table

$$\text{digit}(a \times b) = \text{digit}(a) + \text{digit}(b)$$

This is true even when $a = b$. For instance, digit(10)=1 and digit(10$\times$ 10) = 2.

Consider the question what is digit(316.2278)? That seems a odd question unless you realize that $316.2278 \times 316.2278 = 100,000$. Since digit(100000) = 5, it must be that digit(316.2278) = 5/2.

Another question: what is digit(17.7828)? This seems crazy, until you notice that $17.7828^2 = 316.2278$. So digit(17.78279) = 5/4.

For a couple of thousand years mathematicians have known how to compute the square root of any number to a high precision. By taking square roots and dividing by two, it's easy to fill in more rows in the digit()-function table. You get even more rows by noticing other simple patterns like

$$\text{digit}(a/10) = \text{digit}(a) - 1 \quad \text{and} \quad \text{digit}(10a) = \text{digit}(a) + 1$$

Here are some additional rows in the table

input	output	Why?
316.2278	2.5	From $\sqrt{100,000}$
17.17828	1.25	From $\sqrt{316.2278}$
4.21696	0.625	From $\sqrt{17.17828}$
31.62278	1.5	From 316.2278/10
3.162279	0.5	From 31.62278/10

You can play this game for weeks. We asked the computer to play the game for about half a second and expanded the original

digit() table to 7975 rows.

Figure @ref(fig:expanded-log) plots the expanded digits() function table.

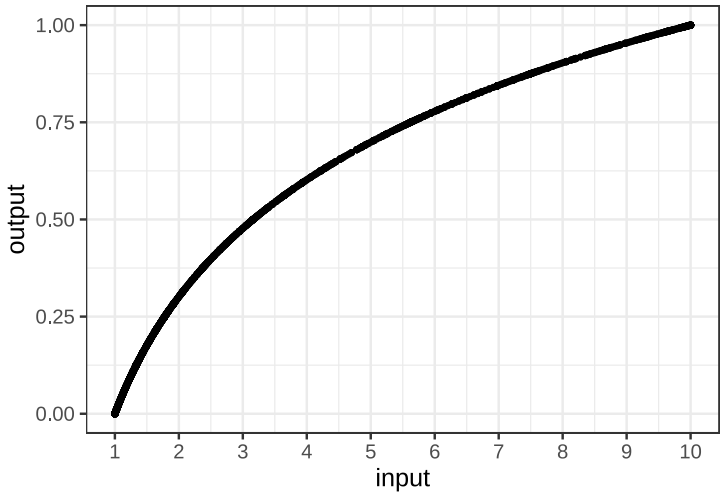
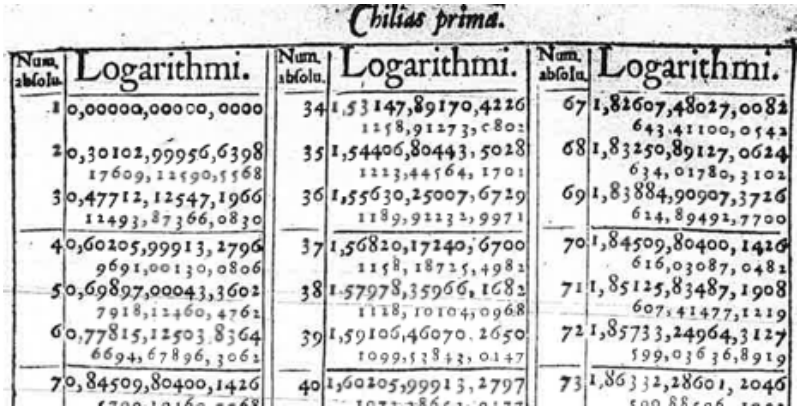


Figure 14.10: The digit function with more entries

Now we have a smooth function that plays by the digit rules of multiplication.

Henry Briggs and his assistants did a similar calculation by hand. Their work was published in 1617 as a table.


Chilias prima.

Num. absolu.	Logarithmi.	Num. absolu.	Logarithmi.	Num. absolu.	Logarithmi.
1	0,00000,00000,0000	34	1,53147,89170,4226	67	1,82607,48027,0082
			1258,91273,c802		643,41100,0542
2	0,30102,99956,6398	35	1,54406,80443,5028	68	1,83250,89127,0624
	17609,12590,5568		1213,44564,1701		634,01780,3102
3	0,47712,12547,1966	36	1,55630,25007,6729	69	1,83884,90907,3726
	12493,87366,0830		1189,92232,9971		614,89492,7700
4	0,60205,99913,2796	37	1,56820,17240,6700	70	1,84509,80400,1426
	9691,00130,0806		1158,18725,4982		616,03087,0482
5	0,69897,00043,3602	38	1,57978,35966,1682	71	1,85125,83487,1908
	7918,12460,4761		1128,10104,0968		607,41477,1219
6	0,77815,12503,8364	39	1,59106,46070,2650	72	1,85733,24964,3127
	6694,67896,3062		1099,53843,0147		599,03636,8919
7	0,84509,80400,1426	40	1,60205,99913,2797	73	1,86332,28601,2046

Figure 14.11: Part of the first page of Henry Briggs table of logarithms

The table was called the *Chilias prima*, Latin for "First group of one thousand." True to its name, the table gives the output of digits() for the inputs 1, 2, 3, ..., 998, 999, 1000. For instance, as you can see from the top row of the right-most column, digits(67) = 1.82607480270082.

In everyday speech, 67 has two digits. The authors of *Chilias prima* sensibly didn't use the name "digit()" for the function. They chose something more abstract: "logarithm()". Nowadays, this function is named $\log_{10}()$. In R, the function is called `log10()`.

```
log10(67)
## [1] 1.826075
```

Our main use for $\log_{10}()$ (in R: `log10()`) will be to count digits in order to quickly compare the magnitude of numbers. The difference digits(x) - digits(y) tells how many factors of 10 separate the magnitude of the x and y.

Another important logarithmic/digit-counting function is $\log_2()$, written `log2()` in R. This counts how many *__binary digits__ are in a number. For us, $\log_2(x)$ tells how many times we need to double, starting at 1, in order to reach x. For instance, $\log_2(67) = 6.06609$, which indicates that $67 = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2^{0.06609}$

$\log_2(x)$ and $\log_{10}(x)$ are proportional to one another. One way to think of this is that they both count "digits" but report the results in different units, much as you might report a temperature in either Celsius or Fahrenheit. For $\log_2(x)$ the units of output are in ***bits***. For $\log_{10}(x)$ the output is in ***decades***.

A third version of the logarithm function is called the ***natural logarithm*** and is denoted ln() in math notation and simply `log()` in R. We'll need additional calculus concepts before we can understand what justifies calling ln() "natural."

14.8 Exercises

Exercise 15.2: ILXEG unassigned

Exercise 15.4: j3xe unassigned

Exercise 15.5: RWESX unassigned

Exercise 15.6: gmZiWh unassigned

Exercise 15.7: EWLCI unassigned

Exercise 15.8: TLEXE unassigned

Exercise 15.9: SELIX unassigned

```
rr insert_calcZ_exercise("15.10", "PeQJCA", "Exercises/octopus-look-radio.Rmd")
```

```
rr insert_calcZ_exercise("3.2", "pUKm4c", "Exercises/sheep-buy-dress.Rmd")
```

15 Dimensions and units

Next time you're at a family gathering with your 10-year old cousin, give her the following math quiz.

1. What's 3 + 2?
2. What's 7 - 3?
3. What's 3 miles + 2 miles?
4. What's 3 miles + 2 kilometers?
5. What's 3 miles + 2 kilograms?

I don't know your cousin, but I suspect she will have an easy time answering (1) and (2) correctly. As for (3), she might give the correct answer, "5 miles," or just say "5." If so, you'll follow up with "5 what?" at which point she'll definitely say "miles."

(4) is a bit harder. You might need to prompt her with the information that 1 kilometer is about 0.6 miles. Then, if she's pretty smart, she'll answer "4.2 miles."

10-year olds are pretty creative, so I'm not sure how she'll answer (5). But if you ask your Ph.D. aunt, she'll answer along the lines of "silly question," or "there's no such thing." That's true.

Consider these everyday quantities:

i. 60 miles per hour: a typical *speed* for driving on a highway
ii. 2106 square feet: the in-bounds *area* for a court used for singles tennis.
iii. 355 cubic centimeters: the *volume* in a canned beverage (in the US).
iv. 2.5 gallons per minute: the US mandated maximum *flow rate* for water through a shower head.
v. 35 miles per gallon: a typical fuel economy for a small car in the US.

vi. 0.044 lbs per square foot: the *body-mass index* of Dwayne ("The Rock") Johnson. In the more conventional units of kg per square meter, his BMI is 30.8.

Consider how you would measure such things:

i. We ordinarily use a speedometer to measure instantaneous car speed and police use a radar gun. But fundamentally, you measure the distance traveled and the time used and divide distance by time.
ii. Most people would rely on the internet for this information, but you would check your local court by measuring the width (27 feet is the standard) and the length of the course (78). Multiply the two.
iii. Pour the beverage into a measuring cup and read off the volume. But more fundamentally, you could measure the circumference of the can ($2\pi r$), square it ($4\pi^2 r^2$) and divide by 4π to get the cross section area of the can. Then multiply that by the height of the can.
iv. We don't usually monitor water used by a shower. But if you need to, get a 5-gallon pail (the standard volume of the plastic pails used for so many purposes in construction), put it under the shower head, and measure the time it takes to fill the pail. Divide the volume by the time.
v. Record the mileage on the car's odometer when you fill up the car with gas. Drive. When you next get gas, measure the new odometer reading and the volume of gas you purchased. Divide the change in odometer reading by the gas volume. (In Europe, you would divide the gas volume by the change in odometer reading.)
vi. Weigh Dwayne. The scale is usually graduated in both pounds and kilograms: take your choice. Measure his height; the ruler-in-the-doorway method works well. Then divide his weight by the square of his height.

Evidently, it makes sense to *multiply* and *divide* different types of quantities: feet, gallons, kilometers, kilograms, pounds, hours, ….. But you won't ever see a quantity constructed by *adding* or *subtracting* miles and hours or gallons and and square feet. You can square feet and cube centimeters, but can you take the square root of a gallon? Does it make sense to raise 2 to the power of 3 yards?

This section is about the mathematical structure of combining quantities; which kinds of mathematical operations are legitimate and which are not.

15.1 Mathematics of quantity

1. [**Fun-10a**] *Know the definition of a fundamental dimension and the notation for the most common ones (definition page 241-242)*

1. [**Fun-10b**] *Understand how derived dimensions are formed from fundamental dimensions (definition page 241-242)*

1. [**Fun-10c**] *Know that units are ways of measuring dimensions and derived dimensions.*

The first step in understanding the mathematics of quantity is to make an absolute distinction between two concepts that, in everyday life, are used interchangeably: ***dimension*** and ***unit***.

Length is a dimension. Meters is a unit of length. We also measure length in microns, mm, cm, inches, feet, yards, kilometers, and miles, to say nothing of furlongs, fathoms, astronomical units (AU), and parsecs.

Time is a dimension. Seconds is a unit of time. We also measure time in micro-seconds, milli-seconds, minutes, hours, days, weeks, months, years, decades, centuries, millenia.

Mass is a dimension. Kilograms is a unit of mass.

Length, time, and mass are called ***fundamental dimensions***. This is not because length is more important than area or volume. It's because you can construct area and volume by multiplying lengths together. This is evident when you consider units of area like square-inches or cubic centimeters, but obscured in the names of units like acre, liter, gallon.

We use the notation L, T, and M to refer to the fundamental dimensions. (Electrical current Q is also a fundamental dimension, but we won't have much use for it in our examples. Also

useful are Θ ("theta") for temperature, S for money, and P for a count of organisms such as the population of the US or the size of a sheep herd.)

Brackets translate between a quantity and the dimension. For instance, [1 yard] = L, [1000 kg] = M, [3 years] = T, [10 μ (microns)] = L,

15.2 Compound dimensions

There are other dimensions: volume, force, pressure, energy, torque, velocity, acceleration, and such. These are called ***compound dimensions*** because we represent them as combinations of the fundamental dimensions, L, T, and M. The notation for these combinations involves multiplication and division. For instance:

- Volume is $L \times L \times L = L^3$, as in "cubic centimeters"
- Velocity is L/T, as in "miles per hour"
- Force is $M\ L/T^2$, which is obscure unless you remember Newton's Second Law that $F = m\,a$: "force equals mass times acceleration." In terms of dimension, mass is M, acceleration is L/T^2. Multiply the two together and you get the dimension "force."

Multiplication and division are used to construct a compound dimension from the fundamental dimensions L, T, and M.

Addition and subtraction are **never** used to form a compound dimension.

Much of the work in understanding dimensions involves overcoming the looseness of everyday speech. Remember the weight scale graduated in pounds *and* kilograms. The unit kilograms is a way of measuring M, but the unit of pounds is a way of measuring *force*: $M\ L/T^2$.

Weight is not the same as mass. This makes no sense to most people and doesn't really matter in everyday life. It's only when you venture off the surface of the Earth that the difference shows up. The Mars rover Perseverence has a weight of 1000 kg on Earth. It was weightless for most of its journey to Mars.

After landing on Mars, Perseverence weighed just 380 kg. But the rover's mass didn't change at sll.

Another source of confusion carried over from everyday life is that sometimes we measure the same quantity using different dimensions. You can measure a volume by *weighing* water; a gallon of water weighs 8 pounds, a liter of water has a mass of 1 kg. Serious bakers measure flour by weight; a casual baker uses a measuring cup. We can measure water volume with length because water has a (more-or-less) constant mass density. But 8 pounds of gasoline is considerably more than a gallon. It turns out that the density of flour varies substantially depending on how it's packed, on humidity, etc. This is why it matters whether you weigh flour for baking or measure it by volume. You can measure time by the swing of a pendulum. To measure the same time successfully with different pendula they need to have the same length, not the same mass.

A ***unit*** is a conventional amount of a quantity of a given dimension. All lengths are the same dimensionally, but they can be measured with different conventions: inches, yards, meters, … Units for the same dimension can all be converted unambiguously one into the other. A meter is exactly the same quantity of length as 39.3701 inches, a mile is the same length as 1609.34 meters. Liters and gallons are both units of volume (L^3): a gallon is the same as 3.78541 liters.

You will hear it said that a kilogram is 2.2 pounds. That's not strictly true. A kilogram has dimension M and a pound has dimension $\mathrm{ML/T}^2$. Quantities with different dimensions cannot be "equal" or even legitimately compared to one another. Unless you bring something else into the game that physically changes the situation, for instance gravity (dimension of acceleration due to gravity (dimension $\mathrm{L/T}^2$). The *weight* of a kilogram on the surface of the Earth is is 2.2 pounds because gravitational acceleration is (almost) the same everywhere on the surface of the Earth.

It's also potentially confusing that sometimes different dimensions are used to get at the same idea. For instance, the same car that gets 35 miles / gallon in the US (dimension $\mathrm{L/L}^3 = 1/\mathrm{L}^2$) will use 6.7 liters per 100 kilometers ($\mathrm{L}^3/L = \mathrm{L}^2$)

in Europe. Same car. Same fuel. Different conventions using different dimensions.

Keeping track of the various compound dimensions can be tricky. For many people, it's easier to keep track of the physical relationships involved and use that knowledge to put together the dimensions appropriately. Often, the relationship can be described using specific calculus operations, so knowing dimensions and units helps you use calculus successfully.

Easy compound dimensions that you likely already know:

i. $[\boldsymbol{Area}] = \mathrm{L}^2$. Some corresponding units to remind you: "square feet", "square miles", "square centimeters."
ii. $[\boldsymbol{Volume}] = \mathrm{L}^3$. Units to remind you: "cubic centimeters", "cubic feet", "cubic yards." (What landscapers informally call a "yard," for instance "10 yards of topsoil" should properly be called "10 cubic-yards of topsoil.")
iii. $[\boldsymbol{Velocity}] = \mathrm{L/T}$. Units: "miles per hour," "inches per second."
iv. $[\boldsymbol{Momentum}] = \mathrm{ML/T}$. Units: "kilogram meters per second."

Anticipating that you will return to this section for reference, we've also added some dimensions that can be understood through the relevant calculus operations.

- $[\boldsymbol{Acceleration}] = \mathrm{L/T}^2$. Units: "meters per second squared," In calculus, acceleration is the derivative of velocity with respect to time, or, equivalently, the 2nd derivative of position with respect to time.
- $[\boldsymbol{Force}] = \mathrm{ML/T}^2$ In calculus: force is the derivative of momentum with respect to time.
- $[\boldsymbol{Energy}]$ or $[\boldsymbol{Work}] = \mathrm{ML}^2/\mathrm{T}^2$ In calculus, energy is the integral of force with respect to length.
- $[\boldsymbol{Power}] = \mathrm{ML}^2/\mathrm{T}^3$ In the language of calculus, power is the derivative of energy with respect to time.

Math in the world ...

Density sounds like a specific concept, but there are many different kinds of densities. These have in common that they are a ratio of a physical amount to a geometric extent:

i. a physical amount: which might be mass, charge, people, etc.
ii. a geometric extent: which might be length, area, or volume.

Some examples:

- "paper weight" is the mass per area, typically grams-per-square-meter
- "charge density" is the amount of electrical charge, usually per area or volume
- "lineal density of red blood cells" is the number of cells in a capillary divided by the length of the capillary. (Capillaries are narrow. Red blood cells go through one after the other.)
- "population density" is people per area of ground.

Math in the world ...

The theory of dimensions and units was developed for the physical sciences. Consequently the fundamental dimensions are those of physics: length, mass, time, electrical current, luminous intensity.

Since proper use of units is important even outside the physical sciences, it's helpful to recognize the dimension of several other kinds of quantity.

- "people" / "passengers" / "customers" / "patients" / "cases" / "passenger deaths": these are different different ways to refer to people. We'll consider such quantities to have dimension P, for population.

- "money": Units are dollars (in many varieties: US, Canadian, Australian, New Zealand), euros, yuan (synonym: renminbi), yen, pounds (many varieties: UK, Egypt, Syria, Lebanon, Sudan and South Sudan), pesos (many varieties), dinar, franc (Swiss, CFA), rand, riyal, rupee, won, and many others. Conversion rates depend on situation and national policy, but we will consider money a dimension, denoted by S (from the name of the first coinage, the Mesopotanian Shekel).

Examples:

- Passenger-miles is a standard unit of transport.
- Passenger-miles-per-dollar is an appropriate unit of the economic efficiency of transport.
- Passenger-deaths per million passenger-mile is one way to describe the risk of transport.

15.3 Arithmetic with dimensions

Recall the rules for arithmetic dimensioned quantities. We restate them briefly with the square-bracket notation for "the dimension of." For instance, "the dimension of b" is written $[b]$. We also write $[1]$ to stand for the dimension of a pure number, that is, a quantity without dimension.

Operation	Result	Only if satisfies	Metaphor
Multiplication	$[a \times b] = [a] \times [b]$	anything goes	promiscuous
Division	$[a \div b] = [a] \div [b]$	anything goes	promiscuous
Addition	$[a + b] = [a]$	$[a] = [b]$	monogomous
Subtraction	$[a - b] = [a]$	$[a] = [b]$	monogomous
Trigonometric	$[\sin(a)] = [1]$	$[a] = [1]$	celibate

Operation	Result	Only if satisfies	Metaphor
Exponential	$[e^a] = [1]$	$[a] = [1]$ (of course, $[e] = [1]$)	celibate
Power-law	$[b^a] = \underbrace{[b] \times [b] \times \ldots \times [b]}_{a \text{ times}}$	$[a] = [1]$ with a an integer	exponent celibate
Square root	$[\sqrt{b}] = [c]$	$[b] = [c \times c]$	idiosyncratic
Cube root	$[\sqrt[3]{b}] = [c]$	$[b] = [c \times c \times c]$	idiosyncratic
Hump	$[\text{hump}(a)] = [1]$	$[a] = [1]$	celibate
Sigmoidal	$[\text{sigmoid}(a)] = [1]$	$[a] = [1]$	celibate

15.4 Example: Dimensional analysis

We want to relate the **period** (in T) of a pendulum to it's **length** and **mass**. Acceleration due to **gravity** also plays a role; that has dimension $\text{L} \cdot \text{T}^{-2}$. For simplicity, we'll assume that only the bob at the end of the pendulum cable or rod has mass.

The analysis strategy is to combine the four quantities we think play a role into one total quantity that is **dimensionless**. Since it is dimensionless, it can be constant regardless of the mass, length, period, gravity of each individual situation.

$$[\text{Period}]^a \cdot [\text{Mass}]^b \cdot [\text{Length}]^c \cdot [\text{Gravity}]^d = T^a \cdot M^b \cdot L^c \cdot L^d \cdot T^{-2d} = [1]$$

To be dimensionless:

- $c = -d$, cancel out the L
- $a = 2d$, cancel out the T
- $b = 0$, there's no other mass term, and we need to cancel out the M

All of the exponents can be put in terms of d. That doesn't tell us what d should be, but whatever value for d we decide to choose, we get a ratio that's equivalent to:

$$\frac{[\text{Gravity}] \cdot [\text{Period}]^2}{[\text{Length}]} = [1]$$

This is a relationship between *dimensions* of quantities. To render it into a formula involving the quantities themselves we need to take into account the units.

$$\frac{\text{Gravity} \cdot \text{Period}^2}{\text{Length}} = B$$

We can experimentally determine the numerical value of the dimensionless constant B by measuring the period and length of a pendulum and (on Earth) recognizing that gravitational acceleration on Earth's surface is 9.8 meters-per-second-squared. Such experiment and mathematical models using differential equations give $B = (2\pi)^2$.

15.5 Conversion: Flavors of 1

Numbers are dimensionless but not necessarily unitless. Failure to accept this distinction is one of the prime reasons people have trouble figuring out how to convert from one unit to another.

The number one is a favorite of elementary school students because its multiplication and division tables are completely simple. Anything times one, or anything divided by one, is simply that thing. Addition and subtraction are pretty simple, too, a matter of counting up or down.

When it comes to ***quantities***, there's not just one one but many. And often they look nothing like the numeral 1. Some examples of 1 as a quantity:

- $\frac{180}{\pi} \frac{\text{degrees}}{\text{radians}}$
- $0.621371 \frac{\text{mile}}{\text{kilometer}}$

- $3.78541 \frac{\text{liter}}{\text{gallon}}$
- $\frac{9}{5} \frac{^\circ F}{^\circ C}$
- $\frac{1}{12} \frac{\text{dozen}}{\text{item}}$

I like to call these and others different ***flavors of one***.

In every one of the above examples, the dimension of the numerator matches the dimension of the denominator. The same is true when comparing feet and meters ([feet / meter] is L/L = [1]), or comparing cups and pints ([cups / pint] is $L^3/L^3 = [1]$) or comparing miles per hour and feet per second ([miles/hour / ft per sec] = L/T / L/T = [1]). Each of these quantities has *units* but it has no *dimension*.

It's helpful to think about conversion between units as a matter of multiplying by the appropriate flavor of 1. Such conversion will not change the dimension of the quantity but will render it in new units.

Example: Convert 100 feet-per-second into miles-per-hour. First, write the quantity to be converted as a fraction and alongside it, write the desired units after the conversion. In this case that will be

$$100 \frac{\text{feet}}{\text{second}} \quad \text{into} \quad \frac{\text{miles}}{\text{hour}}$$

First, we'll change feet into miles. This can be accomplished by multiplying by the flavor of one that has units miles-per-foot. Second, we'll change seconds into hours. Again, a flavor of 1 is involved.

What number will give a flavor of one? One mile is 5280 feet, so

$$\frac{1}{5280} \frac{\text{miles}}{\text{foot}}$$

is a flavor of one.

Next, we need a flavor of one that will turn $\frac{1}{\text{second}}$ into $\frac{1}{\text{hour}}$. We can make use of a minute being 60 seconds, and an hour being 60 minutes.

$$\underbrace{\frac{60 \text{ s}}{\text{minute}}}_{\text{flavor of 1}} \underbrace{\frac{60 \text{ minutes}}{\text{hour}}}_{\text{flavor of 1}} = \underbrace{3600 \frac{\text{s}}{\text{hour}}}_{\text{flavor of 1}}$$

Multiplying our carefully selected flavors of one by the initial quantity, we get

$$\underbrace{\frac{1}{5280}\frac{\text{mile}}{\text{foot}}}_{\text{flavor of 1}} \times \underbrace{3600\frac{\text{s}}{\text{hour}}}_{\text{flavor of 1}} \times \underbrace{100\frac{\text{feet}}{\text{s}}}_{\text{original quantity}} = 100\frac{3600}{5280}\frac{\text{miles}}{\text{hour}} = 68.18\frac{\text{miles}}{\text{hour}}$$

15.6 Dimensions and linear combinations

Low-order polynomials are a useful way of constructing model functions. For instance, suppose we want a model of the yield of corn in a field per inch of rain over the growing season, will call it corn(rain). The output will have units of bushels (of corn). The input will have units of inches (of rain). A second-order polynomial will be appropriate for reasons to be discussed in Chapter @ref(optim-and-shape).

$$\text{corn(rain)} \equiv a_0 + a_1\,\text{rain} + \frac{1}{2}a_2\,\text{rain}^2$$

Of course, the addition in the linear combination will only make sense if all three terms a_0, $a_1\,\text{rain}$, and $\frac{1}{2}\,a_2\,\text{rain}^2/2$ have the same dimension. But clearly $[\text{rain}] \neq [\text{rain}^2]$. In order for things to work out, the **coefficients must themselves have dimension**. We know the output of the function will have dimension $[\text{volume}] = \text{L}^3$. Thus, $[a_0] = \text{L}^3$.

$[a_1]$ must be different, because it has to combine with the $[\text{rain}] = \text{L}$ and produce L^3. Thus, $[a_1] = \text{L}^2$.

Finally, $[a_2] = \text{L}$. Multiplying that by $[\text{rain}]^2$ will give the required L^3

MATH IN THE WORLD ...

In everyday communication as well as in most domains such as construction, geography, navigation, and astronomy we measure angles in ***degrees***. 90 degrees is a right angle. But in mathematics, the unit of angle is ***radians*** where a right angle

is 1.5708 radians. (1.5708 is the decimal version of $\pi/2$.) The conversion function, which we'll call raddeg(), is

$$\text{raddeg}(r) \equiv \frac{180}{\pi} r$$

The function that converts degrees to radians, which we'll call degrad() is very similar:

$$\text{degrad}(d) \equiv \frac{\pi}{180} d$$

(Incidentally, $\frac{180}{\pi} = 57.296$ while $\frac{\pi}{180} = 0.017453$.)

In traditional notation, the trigonometric functions such as sin() and tan() can be written with an argument either in degrees or radians. For instance, $\sin(90^\circ) = \sin\left(\frac{\pi}{2}\right)$. Similarly, for the inverse functions like arccos() the units of the output are not specified. This works because there is always a human to intervene between the written expression and the eventual computation.

In R, as in many other computer languages like Python or spreadsheet packages, there is no valid expression like `sin(90 deg)`. In these languages, `90 deg` is not a valid expression (although it might be good if it were valid!). In these and many other languages, angles are always given in radians. Such consistency is admirable, but people are not always so consistent. It is a common source of computer bugs that angles in degrees are handed off to functions like sin() and that the output of arccos() is (wrongly) interpreted as degrees rather than radians.

Function composition to the rescue!

Consider this function given in the Wikipedia article on the position of the sun as seen from Earth.[1]

$$\delta_\odot(n) \equiv -23.44^\circ \cdot \cos\left[\frac{360^\circ}{365\,\text{days}} \cdot (n + 10)\right]$$

Where n is zero at the midnight marking New Years and increases by 1 per day. (The $n + 10$ has units of days and translates New Years back 10 days, to the day of the winter solstice.)

[1] Article accessed on May 30, 2021

$\delta_{\odot}()$ gives the declination of the sun: the latitude pieced by an imagined line connecting the centers of the earth and the sun.

The Wikipedia formula is well written in that it uses some familiar numbers to help the reader see where the formula comes from. 365 is recognizably the length of the year in days. 360° is the angle traversed when making a full cycle around a circle. 23.44° is less familiar, but the student of geography might recognize it as the latitude of the Tropic of Cancer, the latitude farthest north where the sun is directly overhead at noon (on the day of the summer solstice).

But there's a world of trouble for programmer who implements the formula as ::: {.cell layout-align="center" fig.showtext='true'}

```
dec_sun <- makeFun(-23.44 * cos((360/365)*(n+10)) ~ n)
```

For instance, the equinoxes are around March 21 (n=81) and Sept 21 (n=264). On an equinox, the delination of the sun is zero degrees. But let's plug $n = 81$ and $n = 264$ into the formula and see what we get. ::: {.cell layout-align="center" fig.showtext='true'}

```
dec_sun(81)
## [1] 5.070321
dec_sun(264)
## [1] -23.38324
```

::: The equinoxes aren't even equal! And they are not close to zero. Does this mean astronomy is wrong?

The Wikipedia formula should have been programmed this way, using 2 π radians instead of 360 degrees in the argument to the cosine function: ::: {.cell layout-align="center" fig.showtext='true'}

```
dec_sun_right <- makeFun(-23.44 * cos(( 2*pi / 365)*(n+10)) ~ n)
dec_sun_right(81)
```

```
## [1] -0.1008749
dec_sun_right(264)
## [1] -0.1008749
```

:::

The deviation of one-tenth of a degree reflects rounding off the time of the equinox to the nearest day. :::

15.7 Exercises

Exercise 16.1: mHACgC unassigned

Exercise 16.2: UGDKY unassigned

```
rr insert_calcZ_exercise(16.3, "DVGKY", "Exercises/dim-params.Rmd")
```

```
rr insert_calcZ_exercise(16.4, "KGYKY", "Exercises/crow-mean-dress.Rmd")
```

```
rr insert_calcZ_exercise(16.5, "JELCI", "Exercises/Boyd-1.Rmd")
```

```
rr insert_calcZ_exercise(16.6, "aNEcW1", "Exercises/snake-leave-candy.Rmd")
```

```
rr insert_calcZ_exercise("16.7", "VA9Dxi", "Exercises/frog-grow-pantry.Rmd")
```

16 Modeling cycle

AN EXAMPLE FROM THE 01-parameters chapter

Math in the world ...

You can see in Figure @ref(fig:covid-exp2) that the exponential function plotted in blue does not align perfectly with the day-by-day data. For instance, during the interval from March 8 to 21, the function output is consistently higher than the data suggest it should be. As part of the ***modeling cycle*** it's important to notice such discrepancies and try to understand them. In this case, it's likely that during the early days of the pandemic the number of reported deaths understated the actual number of COVID-related deaths. This happens because, in the early days of the pandemic, many deaths from COVID were mis-attributed to other causes.

Effective modelers treat models with skepticism. They look for ways in which models fail to capture salient features of the real world. They have an eye out for deviations between what their models show and what they believe they know about the system being modeled. They consider ways in which the models might not serve the purpose for which they were developed.

When modelers spot a failure or deviation or lack of proper utility, they might discard the model but more often they make a series of small adjustments, tuning up the model until is successfully serves the purposes for which it is intended.

Thus, modeling is a cyclic process of creating a model, assessing the model, and revising the model. The process comes to a sort of preliminary end when the model serves its purposes. But even then, models are often revised to check whether the results are sensitive to some factor that was not included or to check whether some component that was deemed essential really is so.

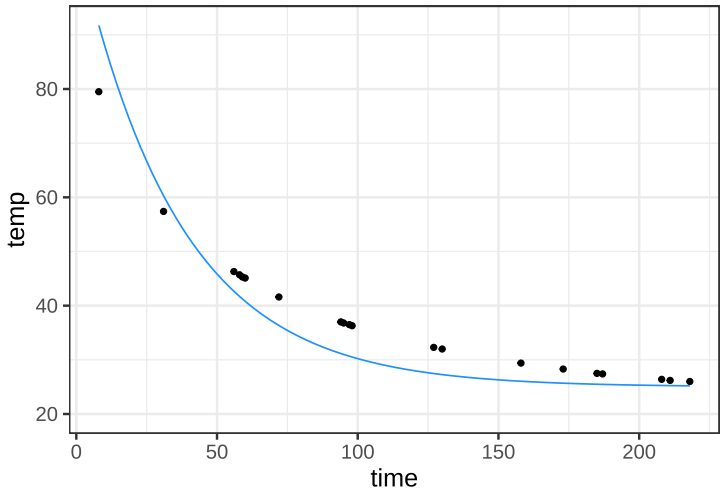
16.1 Example: Cooling water

Looking back on the exponential fitted to the cooling water data in Section @ref(fit-exponential), it looks like our original estimate of the half-life is a bit too small; the data doesn't seem to decay at the rate implied by $k = -0.0277$. Perhaps we should try a somewhat slower decay, say $k = -0.2$ and see if that improves things.

Open an R console and ....

In the cooling water example, we're using only a subset of the data collected by Prof. Wagon. The next commands re-create that subset so that you can work with it. They also plot the data and an exponential model.
::: {.cell layout-align="center" fig.showtext='true'}

```
# reconstruct the sample
set.seed(101)
Stans_data <- CoolingWater %>% sample_n(20)
# Plot the sample and overlay a model
gf_point(temp ~ time, data=Stans_data) %>%
  gf_lims(y = c(20, NA)) %>%
  slice_plot(25 + 83.3*exp(-.0277*time) ~ time, color="dodgerblue")
```



See if $k = -0.02$ provides a better fit to the model. (You can add another `slice_plot()` to be able to compare the original and $k = -0.02$ models.) :::

Later in this course, we'll study ***optimization***. There are optimization techniques for directing the computer to refine the parameters to best match the data. Just to illustrate, here's what we get:

```
refined_params <-
  fitModel(temp ~ A + B*exp(k*time), data = Stans_data,
           start = list(A = 25, B = 83.3, k = -0.0277))
coef(refined_params)
##           A           B           k
## 25.92628463 60.69255269 -0.01892572
new_f <- makeFun(refined_params)
gf_point(temp ~ time, data = Stans_data) %>%
  slice_plot(new_f(time) ~ time, color="dodgerblue")
```

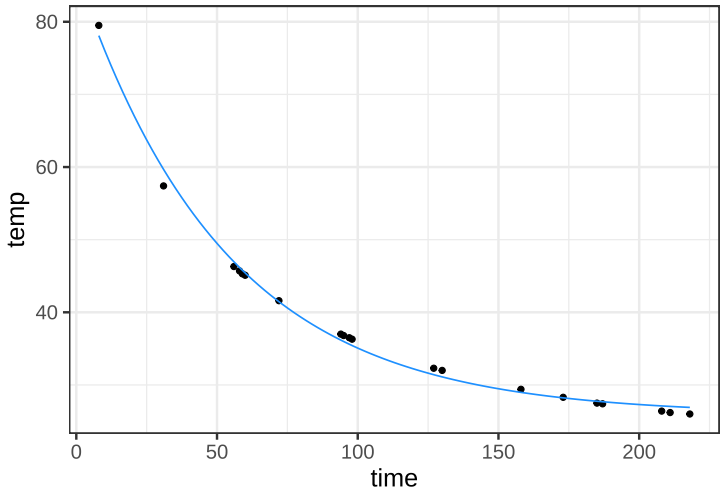


Figure 16.1: Polishing the fit using the rough model as a starting point.

The refined parameters give a much better fit to the data than our original rough estimates by eye.

We had two rounds of the ***modeling cycle***. First, choice of an exponential model and a rough estimate of the parameters A, B, and k. Second, refinement of those parameters using the computer.

Looking at the results of the second round, the experienced modeler can see some disturbing discrepancies. First, the estimated baseline appears to be too high. Related, the initial decay of the model function doesn't seem to be fast enough and the decay of the model function for large t appears to be too slow. Prof. Stan Wagon noticed this. He used additional data to fill in the gaps for small t and refined his model further by changing the basis functions in the linear combination. He hypothesized that there are at least two different cooling processes. First, the newly poured water raises the temperature of the mug itself. Since the water and mug are in direct contact, this is a fast process. Then, the complete water/mug unit comes slowly into equilibrium with the room temperature.

The newly refined model was a even better match to the data. But nothing's perfect and Prof. Wagon saw an opportunity for additional refinement based on the idea that there is a third

physical mechanism of cooling: evaporation from the surface of the hot water. Prof. Wagon's additional circuits of the modeling cycle were appropriate to his purpose, which was to develop a detailed understanding of the process of cooling. For other purposes, such as demonstrating the appropriateness of an exponential process or interpolating between the data points, earlier cycles might have sufficed.

Here's a graph of the model Prof. Wagon constructed to match the data.

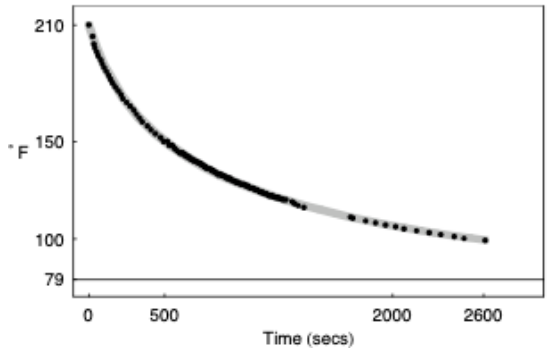


Figure 16.2: A model that combines three exponentials provides an excellent fit.

This is an excellent match to the data. But … matching the data isn't always the only goal of modeling. Prof. Wagon wanted to make sure the model was physically plausible. And looking at the refined parameters, which include two exponential processes with parameters k_1 and k_2, he saw something wrong:

> *But what can we make of k_1, whose [positive value] violates the laws of thermodynamics by suggesting that the water gets hotter by virtue of its presence in the cool air? The most likely problem is that our simple model (the proportionality assumption) is not adequate near the boiling point. There are many complicated factors that affect heat transportation, such as air movement, boundary layer dissipation, and diffusion, and our use of a single linear relationship appears to be inadequate. In the next section [of our paper] we suggest some further experiments, but we also hope that our experiments might*

inspire readers to come up with a better mathematical model.

The modeling cycle can go round and round!

16.2 Example: The tides

In Section @ref(fit-periodic) we looked at a sinusoid model of tide levels in Rhode Island. We left unresolved how to refine the estimate of the period P and find the time offset t_0 in the sinusoidal model

$$\text{tide}(t) \equiv A \sin\left(\frac{2\pi}{P}(t - t_0)\right) + B$$

$$\text{tide}(t) \equiv 1.05 + 0.55 \sin(2\pi(t - t_0)/11)$$

The new parameter, t_0, should be set to be the time of a positive-going crossing of the baseline. Looking at the tide data (black) plotted in Figure @ref(fig:Fun-4-a-3-4) we can pick out such a crossing at about time = 17. Happily, changing the phase does not itself necessitate re-estimating the other parameters: baseline, amplitude, period. This model, incorporating the phase, has been graphed in blue.

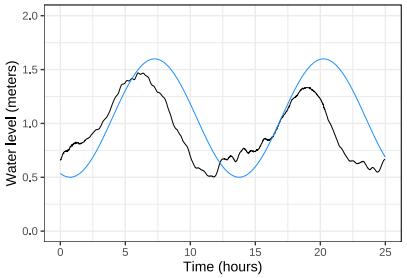


Figure 16.3: Shifting the **phase** of the sinusoid gives the flexibility needed to align the peaks and troughs of the model with the data. Performing this alignment for one peak makes it clear that the period is wrong.

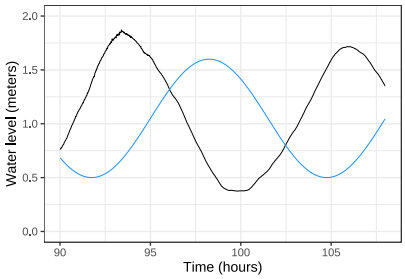


Figure 16.4: Shifting the **phase** of the sinusoid gives the flexibility needed to align the peaks and troughs of the model with the data. Performing this alignment for one peak makes it clear that the period is wrong.

For some modeling purposes, such as prediction of future tides, the phase information is essential. For others, say, description of the amplitude of the tides, not. But getting the phase roughly right can help point out other problems. For instance, in the left panel of Figure @ref(fig:Fun-4-a-3-4) the blue model is roughly aligned with the data. Not at all so in the right panel. What leads to the discrepancy is a bad estimate for the period. 13 hours is roughly right, but over a five-day period the error accumulates until, in the right panel, the model has a trough where the data peak, and *vice versa*.

Although the blue sinusoid is not perfect, having it for comparison suggests that the previously estimated period of 13 hours is too long. We can shorten the period gradually in our model until we find something that better matches the data. For example: Figure @ref(fig:Fun-4-a-3-5) shows that a period of 12.3 hours is a good match to the data.

With this refinement the model is

$$\text{tide}(t) \equiv 1.05 + 0.55 \sin(2\pi(t - 17)/12.3)$$

We might call it quits with the model in Figure @ref(fig:Fun-4-a-3-5). But once we have a pretty good model fit, it's easy to polish the parameter estimates, letting the computer do the tedious work of trying little tweaks to see if it can improve the fit.

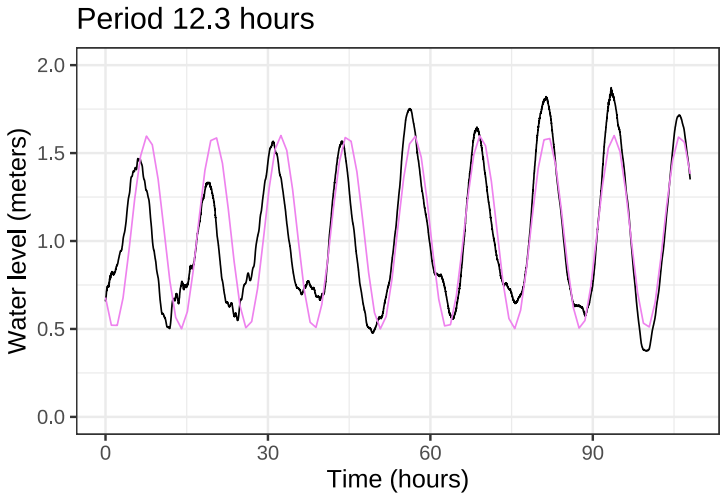


Figure 16.5: With the phase about right, a better estimate can be made of the period: 12.3 hours.

The R/mosaic `fitModel()` can do this tweaking for us. As the following commands show, `fitModel()` takes a tilde expression as input. To the left of the tilde goes the name of the function output in the data frame being used. The right side is a formula for the model, with names used for each parameter and using the names of inputs from the data frame. The second argument is the data frame. The third argument is used to convey an estimate for each parameter; that estimate should be pretty good if `fitModel()` is to be able to refine it.

The output from `fitModel()` is a function, which we're naming `tide_mod()`.

```
tide_mod <-
  fitModel(level ~ A + B*sin(2*pi*(hour-t0)/P),
  data = RI_tide,
  start=list(A=1.05, B=0.55, t0=17, P=12.3))
coef(tide_mod)
##          A          B         t0          P
##  1.0220540  0.4998367 15.3899905 12.5593556
```

The command `coef(tide_mod)` displays the parameters found by `fitModel()` which will be an improvement—perhaps a big

improvement, perhaps not—on our original estimates.

These new parameters differ only slightly from the ones shown in Figure @ref(fig:Fun-4-a-3-5), but the match to the data with the new coefficients is discernably better, even by eye.

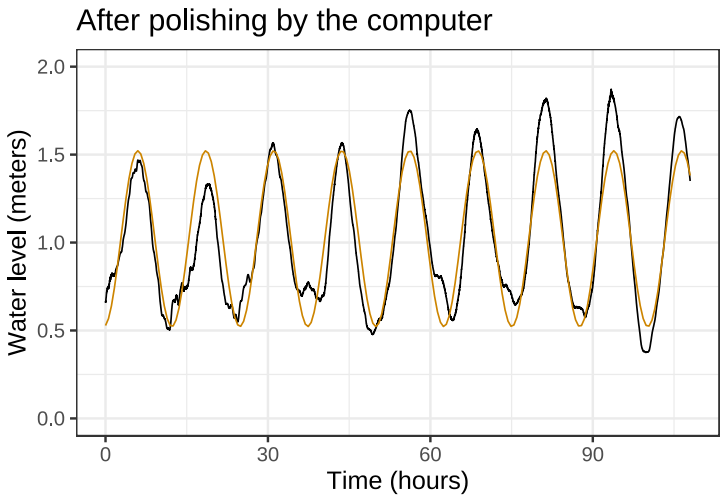


Figure 16.6: Polishing the parameters of the sinusoid

This last model seems capable of making reasonable predictions, so if we collected up-to-date data we might be able to fit a new model to predict the tide level pretty accurately a few days ahead of time. Also, the excellent alignment of the model peaks with the data tell us that the cyclic tide has a period that is constant, at least so far as we can tell.

With the period estimate $P = 12.56$ hours, we can go looking for other phenomena that might account for the tides. The period of the day-night cycle is, of course 24 hours. So the tides in Providence come in and out twice a day. But not exactly. Something else must be going on.

Isaac Newton was the first to propose that the tides were caused by the gravitational attraction of the Moon. A complete cycle of the Moon—moon rise to moon rise—takes about 50 minutes longer than a full day: the Earth revolves once every 24 hours, but in that time the Moon has moved a bit further on in its orbit of the Earth. So the Moon's period, seen from a fixed place on Earth is about 24.8 hours. Half of this, 12.4 hours, is

awfully close to our estimate of the tidal period: 12.56 hours. The difference in periods, 8 minutes a day, might be hard to observe over only 4 days. Maybe with more data we'd get a better match between the tides and the moon.

This is the modeling cycle at work: Propose a model form (a sinusoid), adjust parameters to match what we know (the Providence tide record), compare the model to the data, observe discrepancies, propose a refined model. You can stop the model when it is giving you what you need. The period 12.56 hour model seems good enough to make a prediction of the tide level a few days ahead, and is certainly better than the "two tides a day" model. But our model is not yet able to implicate precisely the Moon's orbit in tidal oscillations.

Discrepancies between a model and data play two roles: they help us decide if the model is fit for the purpose we have in mind and they can point the way to improved models. That the tidal data deviates from the steady amplitude of our model can be a clue for where to look next. It's not always obvious where this will lead.

Historically, careful analysis of tides led to a highly detailed, highly accurate model: a linear combination of sinusoids with periods near a half-day 12.42 , 12.00, 12.66, and 11.97 hours as well components with periods that are about a day long 23.93, 25.82, 24.07, 26.87, and 24.00 hours. A tide-prediction model is constructed by finding the coefficients of the linear combination; these differ from locale to locale. There is no global model of tides, but rather a framework of linear combinations of sinusoids of different periods. What customizes the framework to the tides in a particular locale is the coefficients used in the linear combination.

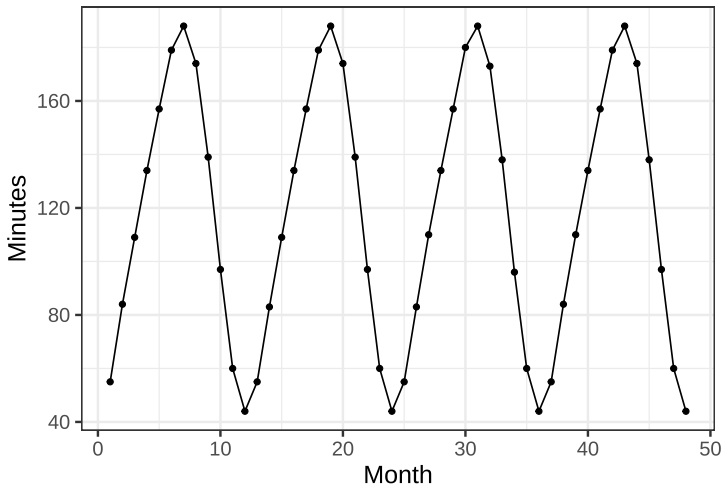
16.3 Modeling project

The data frame `SunsetLA` records the number of minutes after 4 pm until the sun sets in Los Angeles, CA over a 4-year interval from January 2010 (month 1) through December 2013 (month 48).

Open a sandbox and make a plot of sunset time versus month.

Open an R console and ....

```
gf_point(Minutes ~ Month, data = SunsetLA) %>%
  gf_line()
```



We're using both `gf_point()` and `gf_line()`. With data that oscillates up and down, connecting the data points with lines makes it easier to see the pattern.

Question A What is the range of the number of minutes until sunset over the whole 4-year period?

i. 40 to 190 minutesNice!

ii. 120 minutes A range is an interval spanned by *two* numbers.

iii. 40 to 180 minutes The largest values are half a tick mark up from 180. Tick marks are spaced by 20 minutes.

iv. 0 to 48 months That's the *domain*. The *range* is along the vertical axis.

Question B The data fall nicely on a sine-shaped curve. What is the period of that sine?

i. 6 months Look at the number of months from one peak to another.

ii. 11 months Look more carefully. And remember that the change in length of day is an *annual* phenomenon.

iii. 12 monthsGood.

iv. 12 minutes Period refers to an interval on the *domain* of the function, not the range.

The function

$$\text{sunset}(\text{Month}) \equiv A \sin(2\pi\, \text{Month}/12) + C$$

is a *linear combination* of two functions:

1. The constant function `one(Month)`
2. The sine function `sin(2*pi*Month/12)`

The two functions are scaled by C and A, respectively.

Make rough but reasonable numeric estimates for C and A from the data. Then, in the sandbox, define the `sunset()` function using the numerical estimates in the linear combination. Plot your function as a layer on top of the data. (Pipe the `gf_point()` layer to `slice_plot()`.)

Open an R console and ....

```
sunset <- makeFun(A + C*sin(2*pi*(Month - offset)/12),
                  A = __your estimate__,
                  C = __your estimate__,
                  offset=0)
gf_point(Minutes ~ Month, data = SunsetLA) %>%
```

```
    gf_line() %>%
    slice_plot(sunset(Month) ~ Month)
```

The domain for `slice_plot()` is inherited to that implied by the `SunsetLA` data. Notice that the input name in `slice_plot()` corresponds to that established in `gf_point()`.

Question C Your `sunset()` function should be a pretty good match to the data except for one thing. What is that thing?

i. The `sunset()` function has a completely different range than the data. This won't be the case if you have estimated C and A correctly.

ii. The period of the `sunset()` function doesn't match the data. Did you use `sin(2*pi*Month/12)`? If so, the period should be right.

iii. There is a horizontal time shift between `sunset()` and the data.Right!

We're going to fix the problem with `sunset()` by defining a *time offset* to use as a reference. For a sine function, a suitable time offset is the value along the horizontal axis when the phenomenon being modeled crosses C with a positive slope. There are 4 such points along the horizontal axis readily identifiable in the data. (They may not be at an integer value of `Month`.)

Question D Which of these is a suitable value for the time offset?

i. 0 months That's not a time when the data suggest that C is being crossed.

ii. 19 months That's a maximum, not a crossing of C.

iii. 21.5 months That's a crossing of C, but not one with a positive slope.

iv. 15.5 monthsCorrect. This is a good rough value. Since the period is 12 months, you could equally well have said the offset is 3.5 months.

In the original scaffolding, the value of `offset` was zero. Change that to match your answer to the previous question.

Plot out the modified `sunset()` function and confirm that it is a much better match to the data than the original (that is, the one without the time offset). You can "tune" your function by tweaking the numerical values of the A, C, and $offset$ parameters until you get a solid match.

Alternatively, you can use `fitModel()` to do the tuning for you. Plug in your estimates (a.k.a. "guesses") for the parameters in place of the `___` in the following. Then run the code. You'll see your estimate of the function compared to the result of having the computer refine your estimate. Chances are, the computer does a better job of stringing the function through the data.

Open an R console and ....

```
## rough estimates from graph
rough_A <- __estimated_A__
rough_C <- __estimated_C__
rough_offset <- __estimated_offset___
guessed_fun <-
  makeFun(A*sin(2*pi*(Month - offset)/12) + C ~ Month,
          A  = rough_A, C = rough_C,
          offset = rough_offset)

tuned_fun <-
  fitModel(Minutes ~ A*sin(2*pi*(Month - offset)/12) + C,
         data  = SunsetLA,
         start = list(A = rough_A,
                      C = rough_C,
                      offset = rough_offset) )

gf_point(Minutes ~ Month, data = SunsetLA) %>%
```

```
    gf_line(color  = "dodgerblue") %>%
    slice_plot(tuned_fun(Month) ~ Month) %>%
    slice_plot(guessed_fun(Month) ~ Month,  color = "orange3")
```

Perhaps you were expecting the tuned sine function to match the data exactly. It does not. One reason for this is that the Earth's orbit around the Sun is not exactly circular. The sine function is only a *model* of the phenomenon, good for some purposes and not for others. For a more complete explanation, see this article on Wikipedia.

(*Thomas Swalm* contributed to this project.)