MOSAIC Calculus

Daniel Kaplan

4/6/2022

Table of contents

Preface

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: ::: {.cell layout-align="center" fig.showtext='false'}

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

   :::

   - The first command will take about 15 seconds to complete and will display some messages in the Console, which you can ignore. **NOTE**: If you are given

a message asking if you want to install in a "personal" or "private" library, say yes. But if you administer 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 incomprehensible messages will appear in the Console, all of which you can ignore. (If you are 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.)

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: ::: {.cell layout-align="center" fig.showtext='false'}

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

:::

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.

```r
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')
```

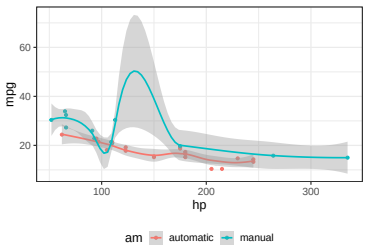


Figure 1: MPG vs horsepower, colored by transmission.

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

---

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.

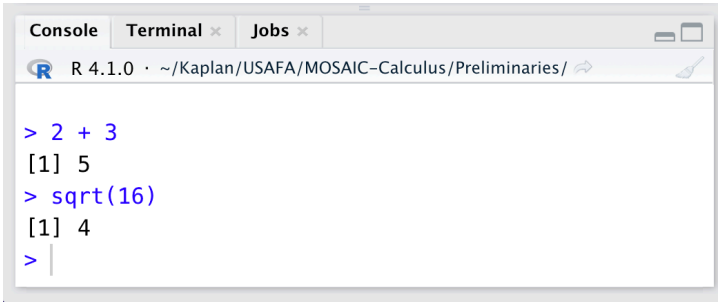


Figure 2.1: An RStudio console tab for writing expressions and evaluating them. The > is the ***prompt*** after which you write your expression, here shown in blue. Pressing the "return" key causes the language interpreter to evaluate the command.

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

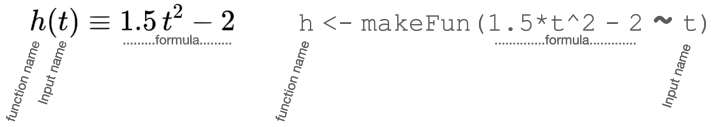
```
h(t=4)
## [1] 22
```

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.

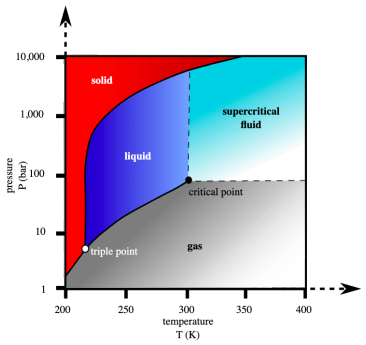


Figure 3.1: Phase diagram for CO_2_. Source Ben Finney, Mark Jacobs, CC0, via Wikimedia Commons

---

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.

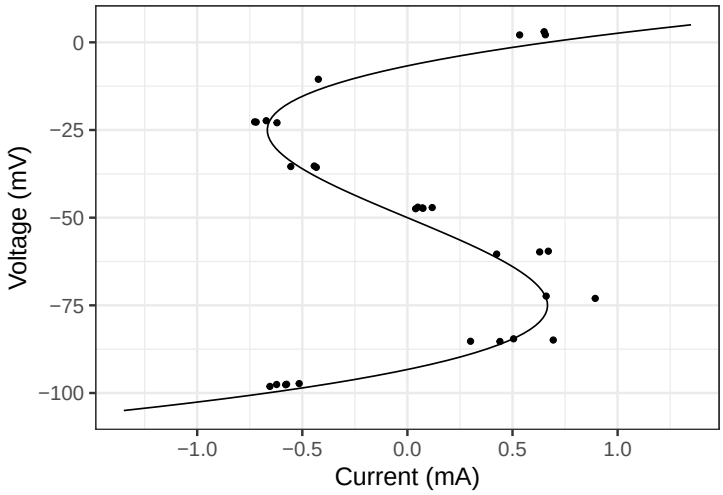


Figure 3.2: Data (simulated) from the squid giant axon experiments. A smooth curve is drawn through the data points.

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))
```

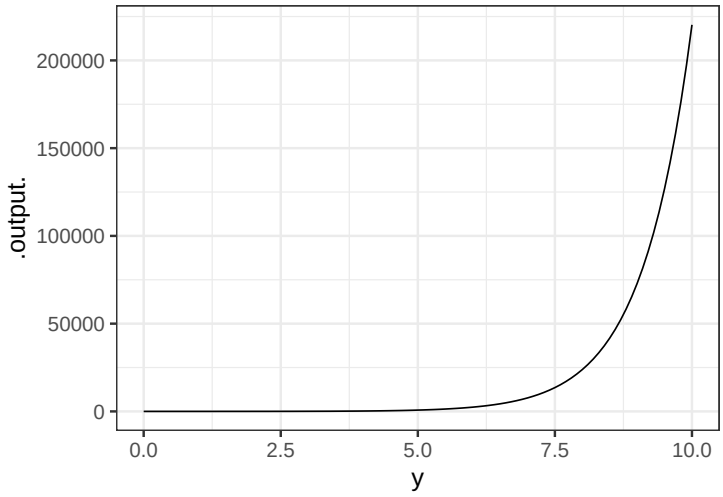


Figure 3.3: Graph of the function $f(y) \equiv ye^y$.

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))
```

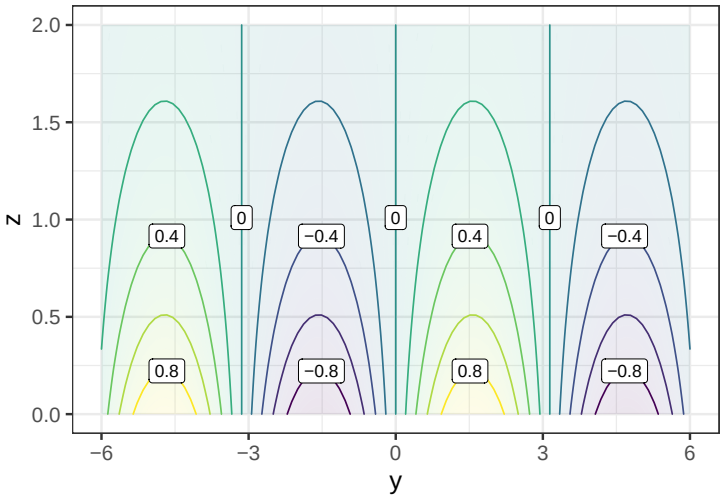


Figure 3.4: Contour plot of a function with two inputs $g(y, z) \equiv e^{-z} \sin(y)$

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))
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is insta
```

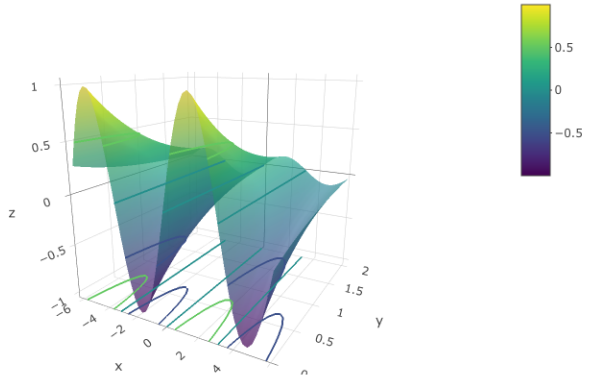


Figure 3.5: Displaying $g(y, z) \equiv e^{-z}\sin(y)$ as a surface plot annotated with contour lines.

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

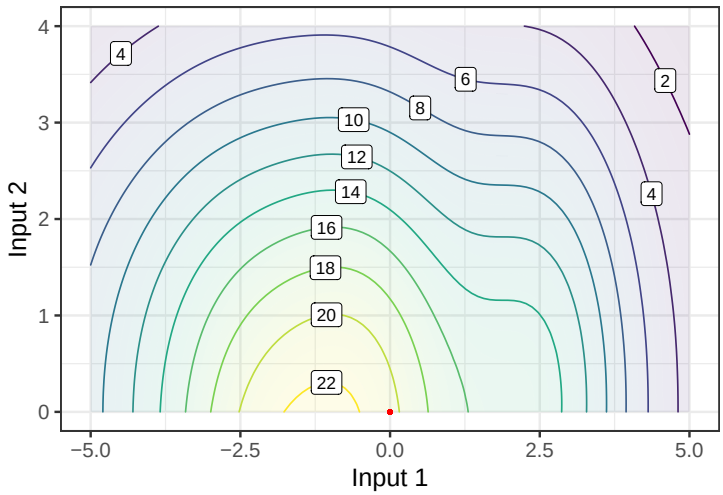
3.2 Interpreting contour plots

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 $(\text{input}_1 = 0, \text{input}_2 = 0)$?



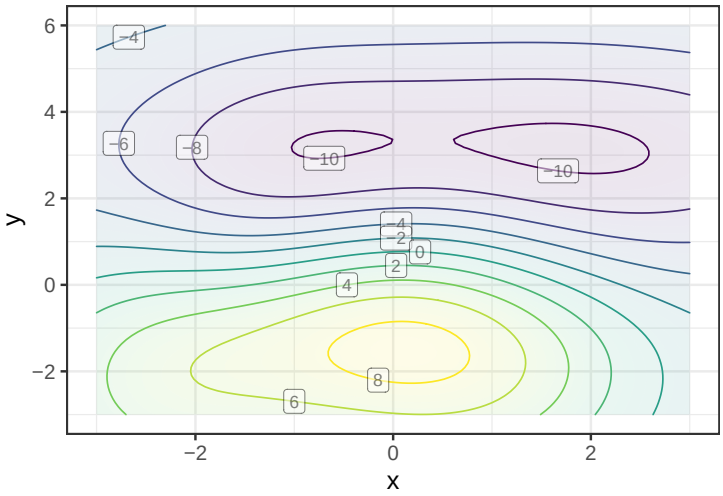
The input pair (0, 0)—which is marked by a small red dot—falls between the contours labeled "20" and "22." This means that the output corresponding to input (0, 0) is somewhere between 20 and 22. The point is much closer to the contour labeled "20", so it's reasonable to see the output value as 20.5.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

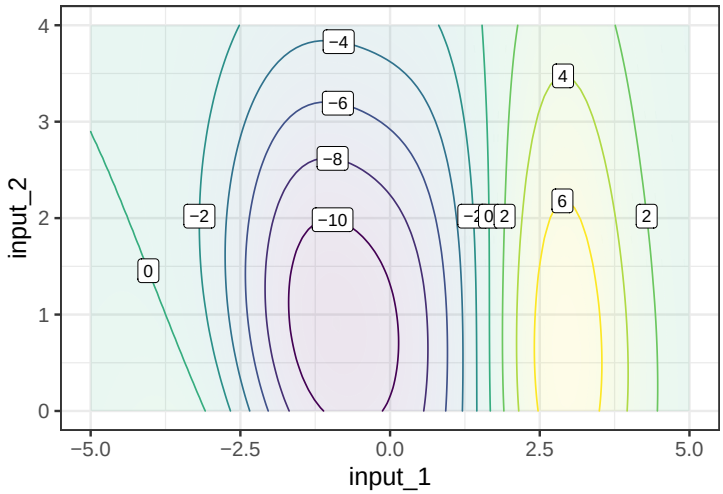


Figure 3.6: A hypothetical relationship among three quantities.

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.

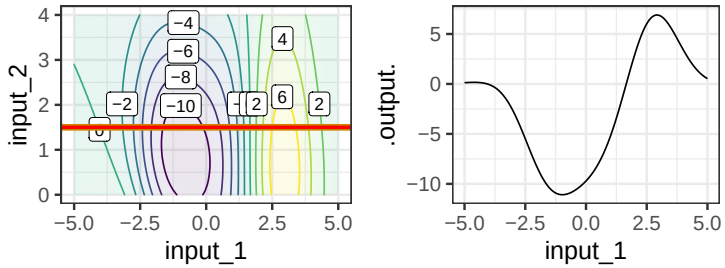


Figure 3.7: Left: A slice (red) through the domain of a contour plot. Right: The value of the function along the red slice presented as a mathematical graph, generated by `slice_plot()`.

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.

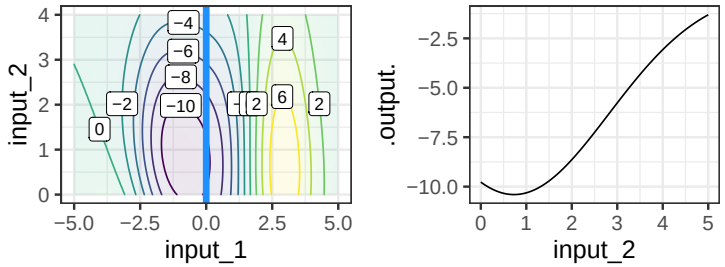


Figure 3.8: A path (blue) along which to cut a slice. The graph is made with `contour_plot()`. Right: Slice plot along the blue path. The graph is made with `slice_plot()`.

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.)

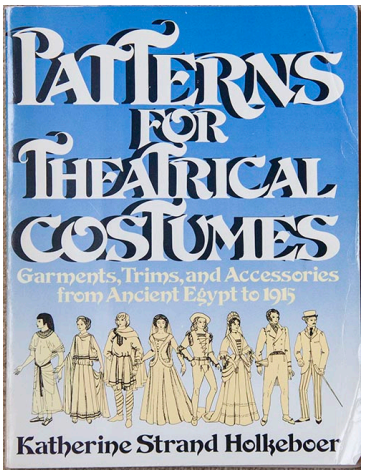


Figure 4.1: A pattern book of theatrical costumes

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."

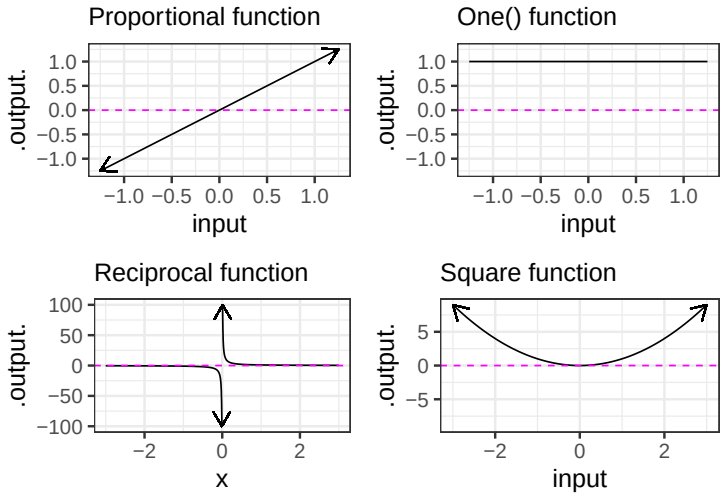
The name "logarithm" is anything but descriptive. The name was coined by the inventor, John Napier (1550-1617), to emphasize the original purpose of his invention: to simplify the work of multiplication and exponentiation. The name comes from the Greek words *logos*, meaning "reasoning" or "reckoning," 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

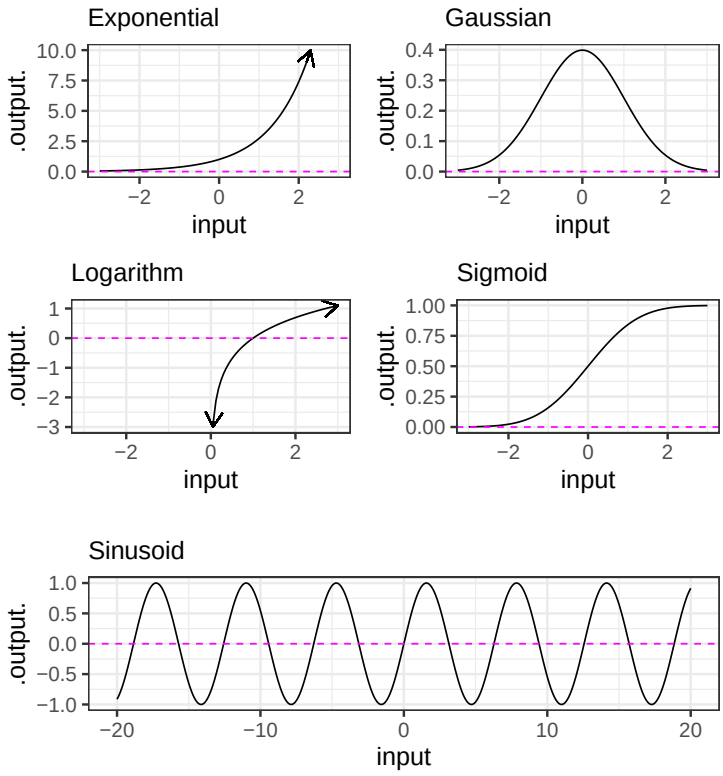
You are going to be building models by selecting an appropriate function or by putting functions together in various ways. This might remind you of Lego blocks. As you know, Legos come in different shapes: 6×2, 4×2, 2×2, and so on. Similarly, each of the pattern-book functions has a distinctively shaped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

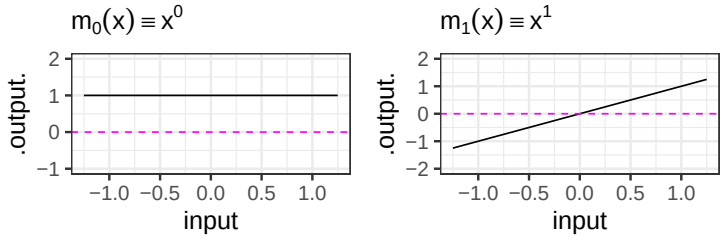
4.2 The power-law family

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

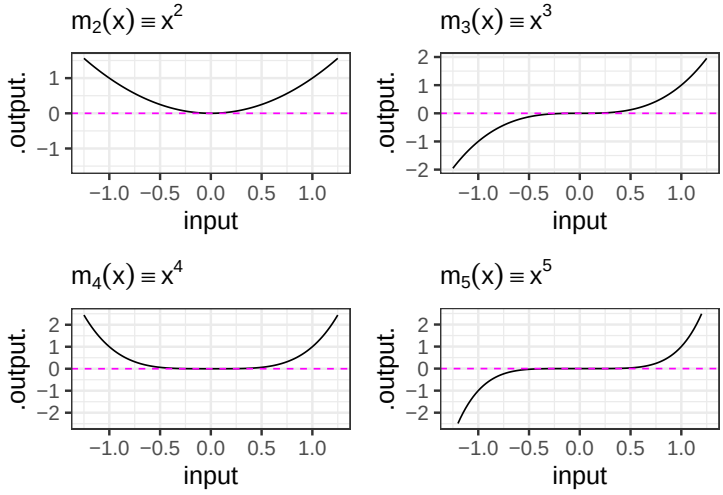


Figure 4.2: Graphs of the monomial functions from order 0 to 5. The dashed magenta line marks zero output.

words, when the input is large (**not** close to zero) the 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

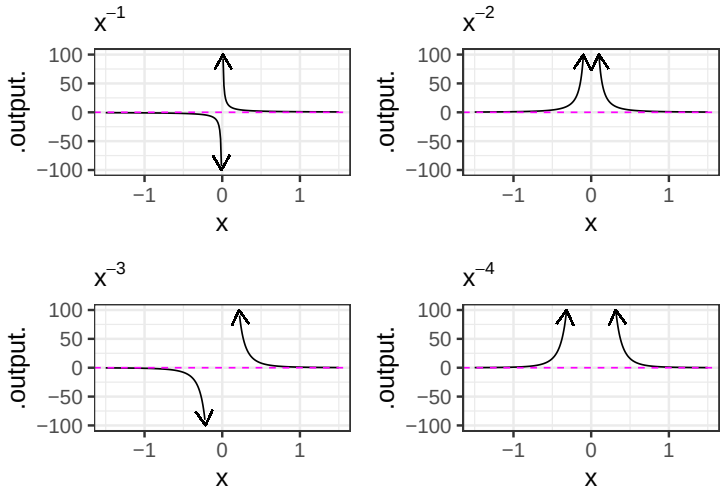


Figure 4.3: Graphs of power-law functions with negative integer exponents. The arrows point to the output being very large when x is near zero.

negative numbers. Computer arithmetic, however, does 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.

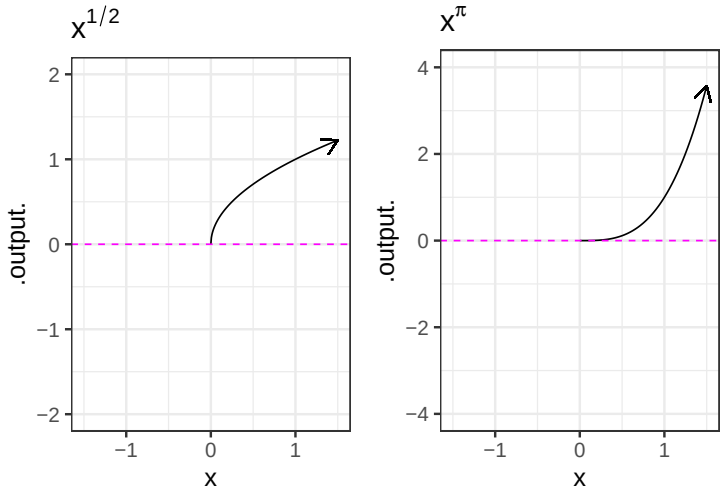


Figure 4.4: The domain of power-law functions with non-integer power is $0 \leq x < \infty$.

- 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

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

$$(\mathbf{ii}) \qquad \left(e^x\right)^y \leftrightarrow e^{xy}$$

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

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

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:

$(\mathbf{v}) \quad \ln(\exp(x)) = x \quad$ 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

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

::: {.takenote data-latex=" "} 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 $x = x_0$: $\text{line}(x = x_0) = a(x - x_0)\,|_{x=x_0} = 0$.

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

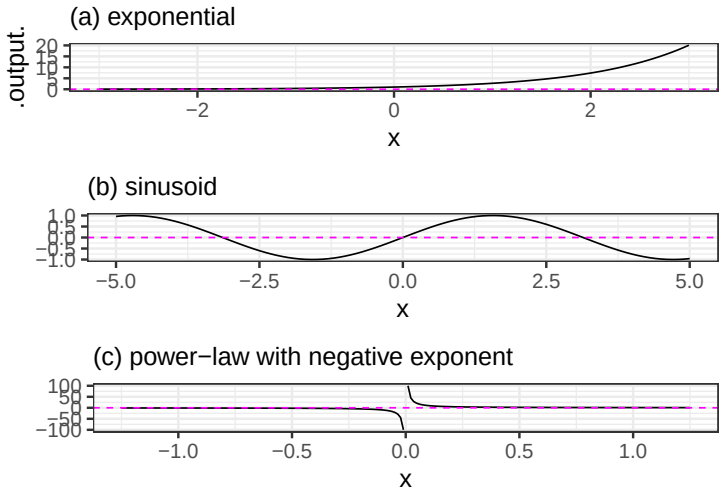


Figure 5.1: Three of the pattern-book functions: (a) exponential, (b) sinusoid, (c) power-law x^{-1}.

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.

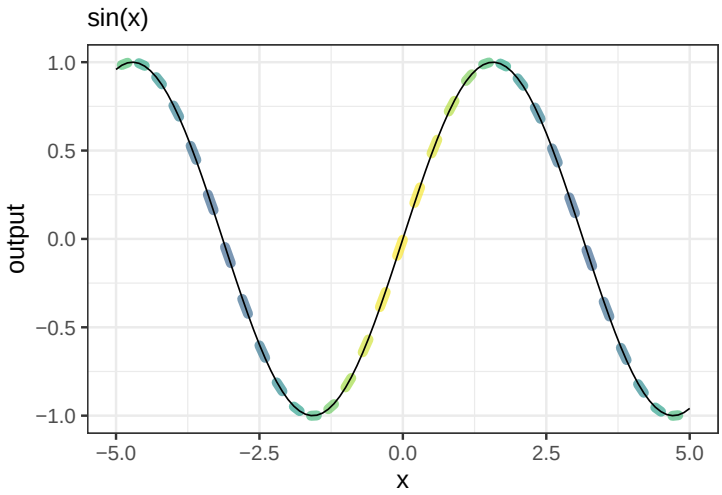


Figure 5.2: Short straight-line segments laid over a graph of the sinusoid. The slope of each line segment is selected to match the local slope of the sinusoid.

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.

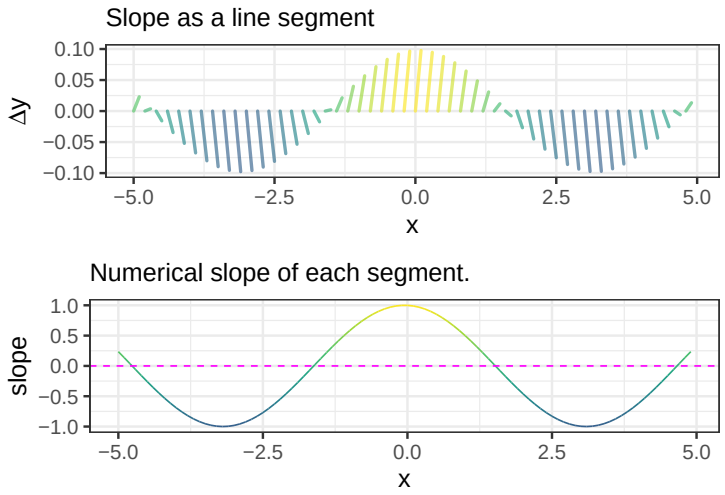


Figure 5.3: Showing just the slope of the sinusoid as a function of input x. Top: representing the slope by the steepness of line segments. Bottom: The numerical value of the slope of each segment.

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.

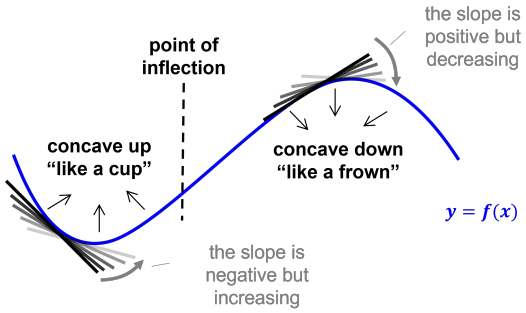


Figure 5.4: A cubic function which is concave up until a point of inflection and concave down thereafter. [Source: Maj. Austin Davis]

"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 $-\infty$.

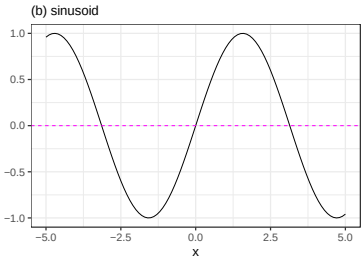
Several of the pattern-book functions have horizontal or vertical asymptotes or both. For instance, the function x^{-1} has a horizontal asymptote of zero for both $x \to \infty$ and $x \to -\infty$.

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.

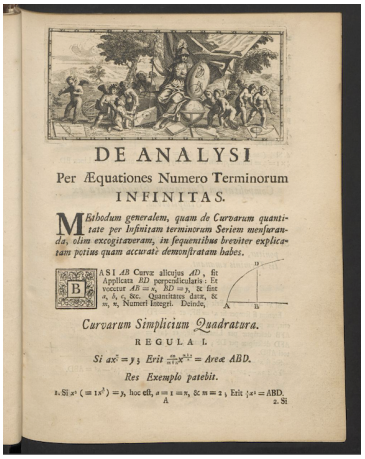

DE ANALYSI
Per Æquationes Numero Terminorum
INFINITAS.
Curvarum Simplicium Quadratura.
REGULA I.
Res Exemplo patebit.

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.

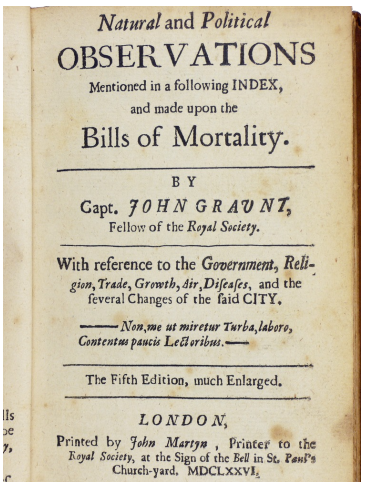

Natural and *Political*
OBSERVATIONS
Mentioned in a following INDEX,
and made upon the
Bills of Mortality.
BY
Capt. JOHN GRAUNT,
Fellow of the *Royal Society*.
With reference to the *Government, Religion, Trade, Growth, Air, Diseases*, and the several Changes of the said CITY.
Non, me ut miretur Turba, 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.

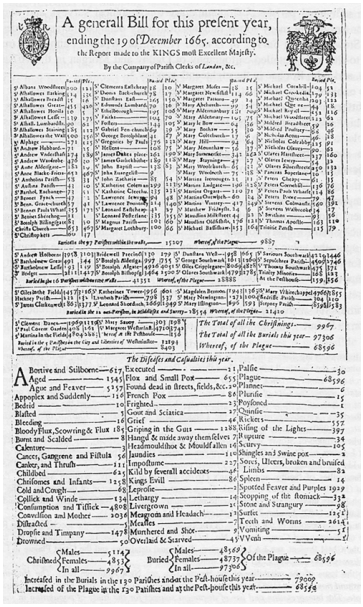
This chapter introduces some pre-calculus basics of working with data which we’ll use extensively in later Blocks of *MOSAIC Calculus*.

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.

The Diseases and Casualties this year.

Abortive and Stillborne	617	Executed	21	Palsie	30
Aged	1545	Flox and Small Pox	655	Plague	68596
Ague and Feaver	5257	Found dead in Streets, fields, &c.	20	Plannet	6
Appoplex and Suddenly	116	French Pox	86	Pleurisie	15
Bedrid	10	Frighted	13	Poysoned	1
Blasted	5	Gout and Sciatica	17	Quinsie	35
Bleeding	16	Grief	44	Rickets	557
Bloody Flux, Scowring & Flux	184	Griping in the Guts	1288	Riding of the Lights	397
Burnt and Scalded	8	Hangd & made away themselves	7	Rupture	34
Calenture	3	Headmouldshot & Mouldfallen	16	Scurvy	105
Cancer, Gangrene and Fistula	56	Jaundies	146	Shingles and Swine pox	8
Canker, and Thrush	111	Impostume	227	Sores, Ulcers, broken and bruised	
Childbed	625	Kild by feverall accidents	56	Limbs	82
Chrisomes and Infants	1258	Kings Evill	86	Spleen	16
Cold and Cough	68	Leprosie	3	Spotted Feaver and Purples	1929
Collick and Wynde	124	Lethargy	20	Stopping of the Stomack	332
Consumption and Tiffick	4808	Livergrown	21	Stone and Strangury	98
Convulsion and Mother	1036	Megrom and Headach	3	Surfeit	1257
Distracted	5	Measles	7	Teeth and Worms	2614
Dropsie and Timpany	1478	Murthered and Shot	7	Vomiting	21
Drowned	50	Overlaid & Starved	45	Wenn	2

Males	51142	Males	48368	Of the Plague	68596
Christned Females	48335	Buried Females	48737		
In all	99657	In all	97306		

Increased in the Burials in the 130 Parishes and at the Pest-house this year 79009.
Increased of the Plague in the 130 Parishes and at the Pest-house this year. 68596

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 Bullingham	1665-06-01	S^{t} Mary le Bow	M	29	plague
Owin Swancott	1665-08-13	Trinitie	M	2	plague
Winifred Romford	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:

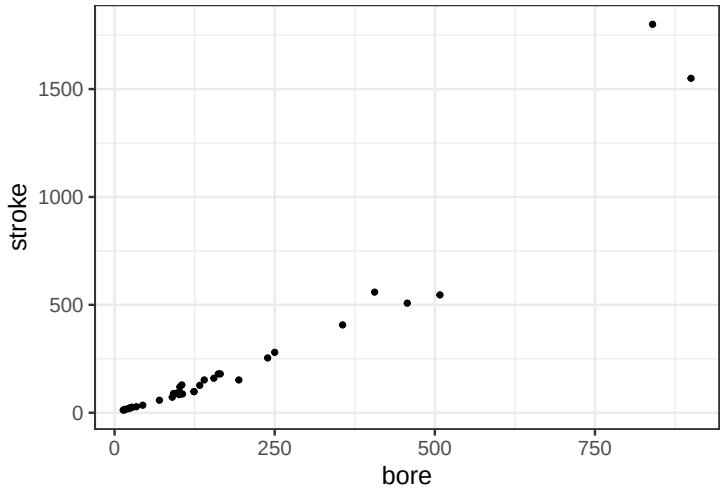


Figure 6.4: A point plot showing the relationship between engine `stroke` and `bore`. Each individual point is one row of the data frame SHOWN IN GIVE TABLE A NAME

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")
```

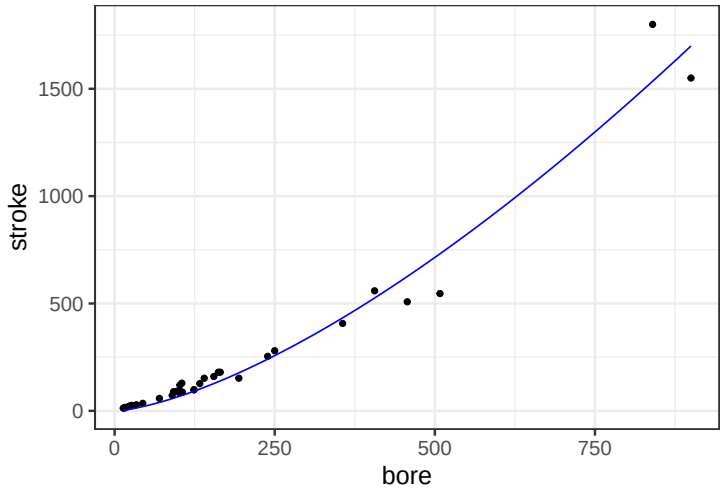
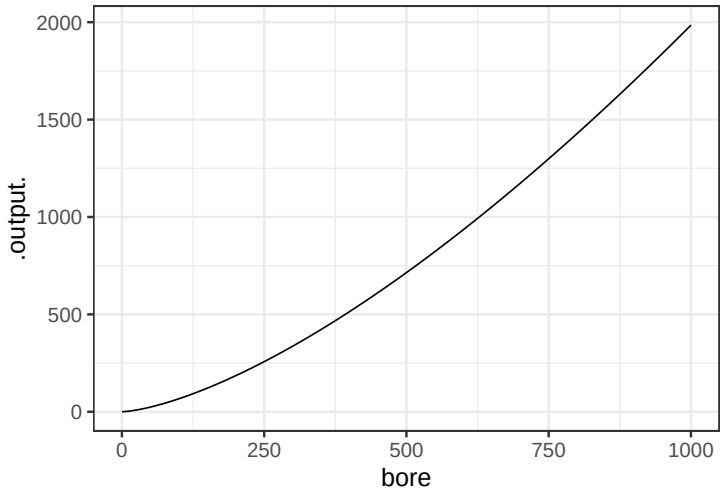


Figure 6.5: A graphic composed of two layers: 1) a point plot; 2) a slice plot of a function fitted to the data in (1).

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

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.

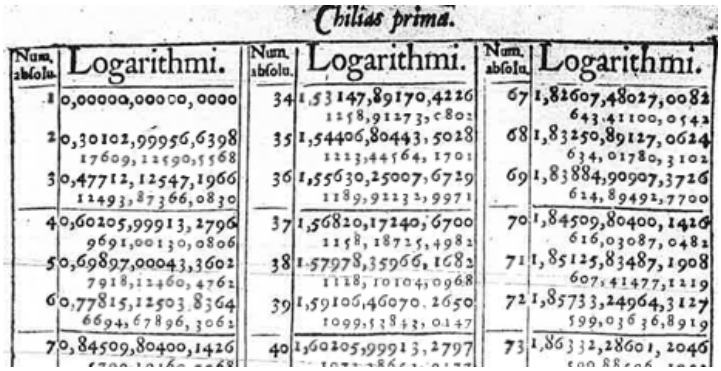


You can do that also when `slice_plot()` is the second layer in a graphics command. But `slice_plot()` can also infer the interval of the domain from previous layers.

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	1,53147,89170,4216 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