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:

```
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 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, which you can ignore. (If you are working from a previous installation of R, you may be asked to update existing packages. Since you can do this at any time in the future, decline the update now.

4. On a daily basis, whenever you need to use the *MOSAIC Calculus* software.
    a. Open the RStudio App in the usual way for your operating system.

   b. In the RStudio console, give these two commands after the console prompt:

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

The 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')
```

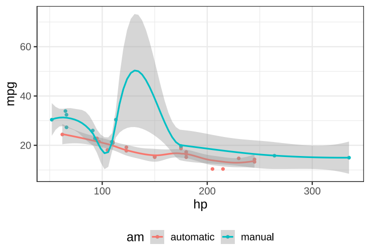


Figure 0.1: MPG vs horsepower, colored by transmission.

Theorem 0.1. *The first theorem is this.*

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

Why did you?

Calling the booboo environment from a div

---

1 Assembling functions

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

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

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

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

where

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

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

1.1 Linear combination

One of the most widely used sorts of combination is called a ***linear combination***. The mathematics of linear combination is, it happens, at the core of the use of math in applications, whether that be constructing a Google-like search engine or analyzing medical data to see if a treatment has a positive effect.

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

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

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

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

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

There are other places where you have seen linear combinations:

- The parameterized ***sinusoid***

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

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

  $$Ae^{kt} + B$$

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

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

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

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

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

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

1.2 Function composition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Math in the world ...

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

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

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

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

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

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

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

A function that relates the cumulative population to the cumulative income is called a ***Lorenz function***. The data are graphed in Figure 1.1 and available as the `US_income` data frame in the SANDBOX. Later, in Figure 1.2, we'll fit parameterized functions to the data.

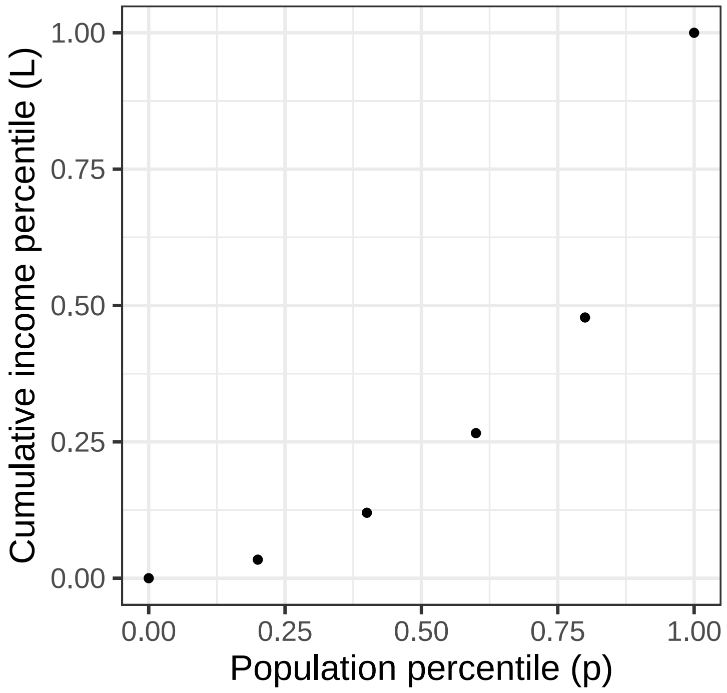


Figure 1.1: Data on household incomes in the US in 2009.

Lorenz curves must:

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

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

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

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

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

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

Figure 1.2 compares the fitted functions to the data.

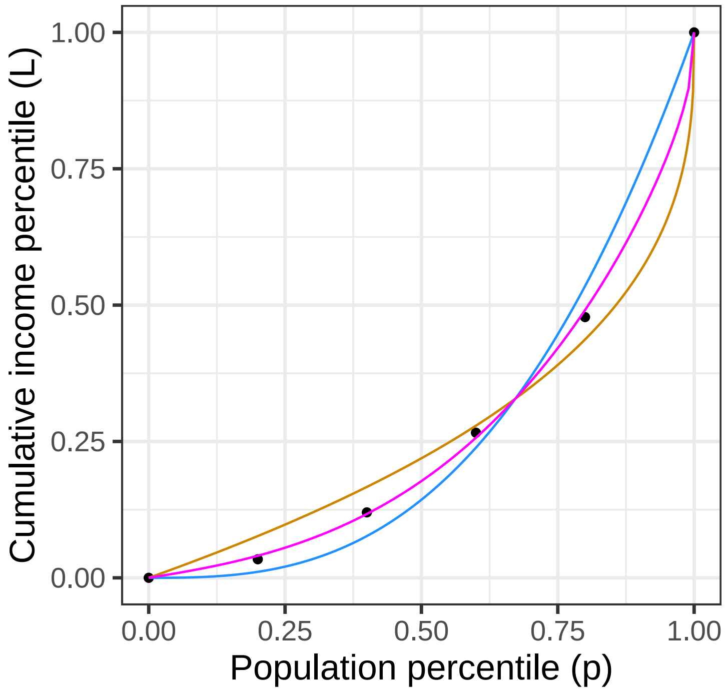


Figure 1.2: Lorenz curves $L_b(p)$ (blue) and $L_q(p)$ (magenta) fitted to the household income data.

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

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

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

- A linear combination of any two Lorenz functions, $aL_1(p) + (1-a)L_2(p)$, so long as the scalars add up to 1, is itself a Lorenz function. For instance, the magenta curve in Figure 1.2 is the linear combination of 0.45 times the tan curve plus 0.55 times the blue curve.

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

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

1.3 Function multiplication

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

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

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

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

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

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

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

For instance ...

Transient vibration

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

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

Function multiplication is used so often in modeling that you'll see it in many modeling situations. Here's one example that is important in physics and communication: the ***wave packet***. Overall, the wave packet is a localized oscillation as in Figure 1.3.

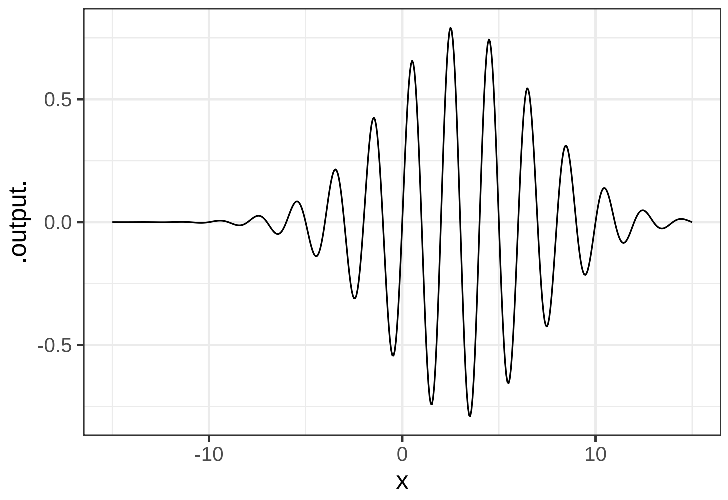


Figure 1.3: A *wave packet* constructed by multiplying a sinusoid and a gaussian function.

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

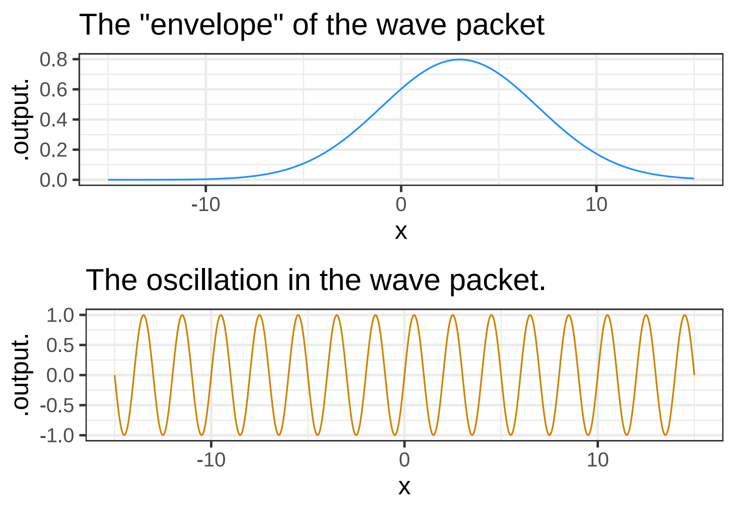


Figure 1.4: The two components of the wave packet in Figure 1.3

MATH IN THE WORLD ...

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

On way to model this pattern is by multiplying a sigmoid by an exponential.(See Figure 1.5.)

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

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

and like this in computer notation:

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

1.4 Watch your domain!

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

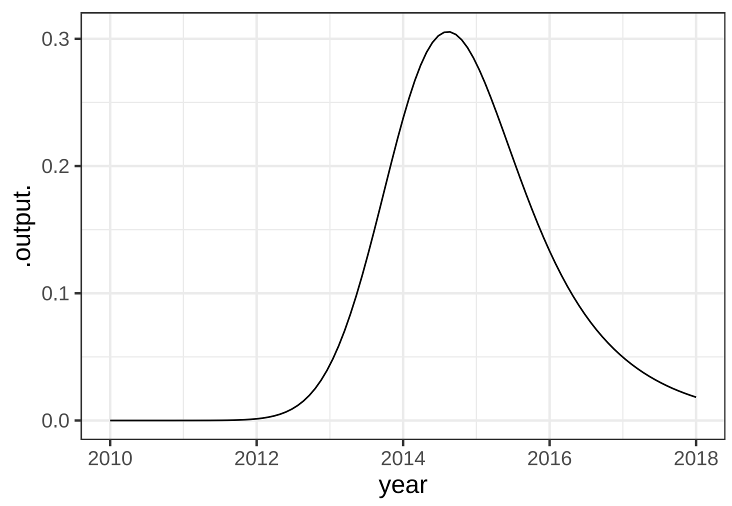


Figure 1.5: Subscriptions to the web messaging service Yik Yak grew exponentially in 2013 and 2014, then collapsed. The company closed in 2017.

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

The two exceptions are:

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

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

R/mosaic commands

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

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

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

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

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

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

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

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

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

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

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

1.5 Splitting the domain

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

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

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

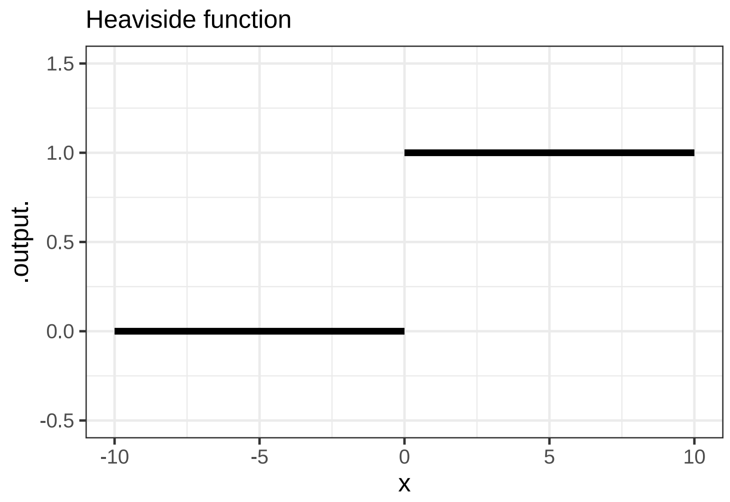
So, instead, let's define $abs()$ in terms of pattern-book functions and scaling. It will look like this: But with the ability to divide the domain into pieces, we gain access to a less mysterious sort of arithmetic operation and can re-write

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

This is an example of a ***piecewise function***, that is a function whose domain is split into two or more intervals and defined by different formulas on those intervals. In the conventional mathematical notation, there is a large { followed by two or more lines. Each line gives a formula for that part of the function and indicates to which interval the formula applies.

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

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



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

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

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

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

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

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

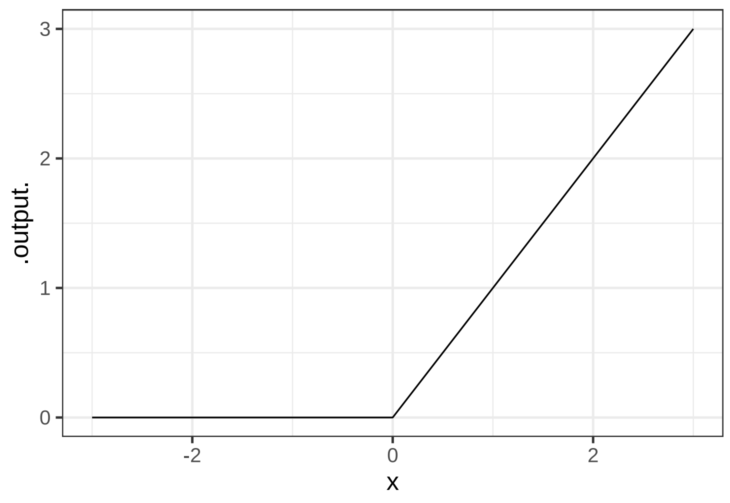
Similarly, the ***ramp function*** is a kind of one-sided absolute value:

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

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

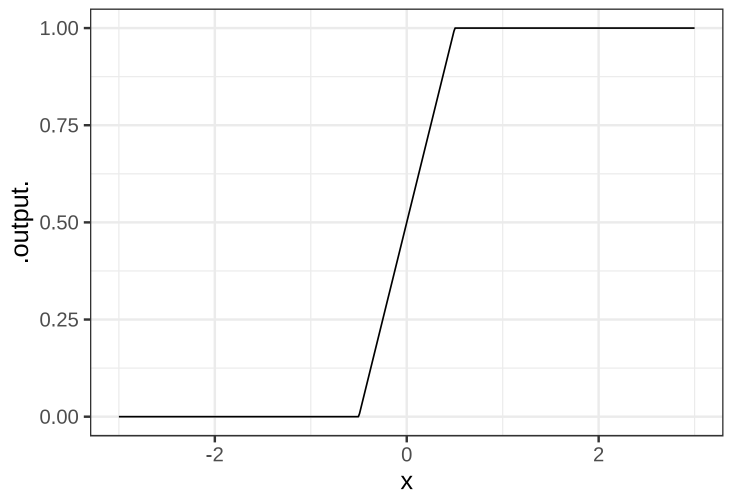
```
ramp <- makeFun(ifelse(0 < x, x, 0) ~ x)
slice_plot(ramp(x) ~ x, domain(x=c(-3, 3)))
```

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



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

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



:::

MATH IN THE WORLD ...

Figure 1.6 is a graph of monthly natural gas use in the author's household versus average temperature during the month. (Natural gas is measured in cubic feet, appreviated *ccf.*)

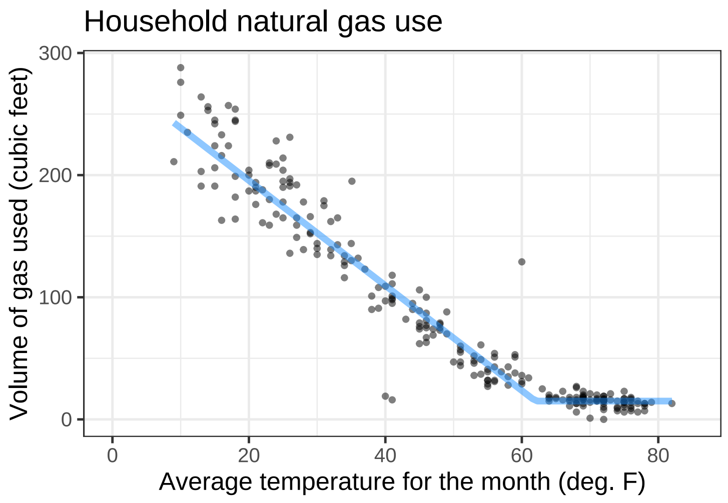


Figure 1.6: The amount of natural gas used for heating the author's home varies with the outside temperature.

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

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

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

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

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

1.6 Exercises

Exercise XX.XX: FCXT1L function composition

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

Exercise XX.XX: HMTG1w function multiplication

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

Exercise 13.05: EDKYV unassigned

Exercise 13.07: E9e7c6 unassigned

Exercise XX.XX: mtHyvJ asymptotes