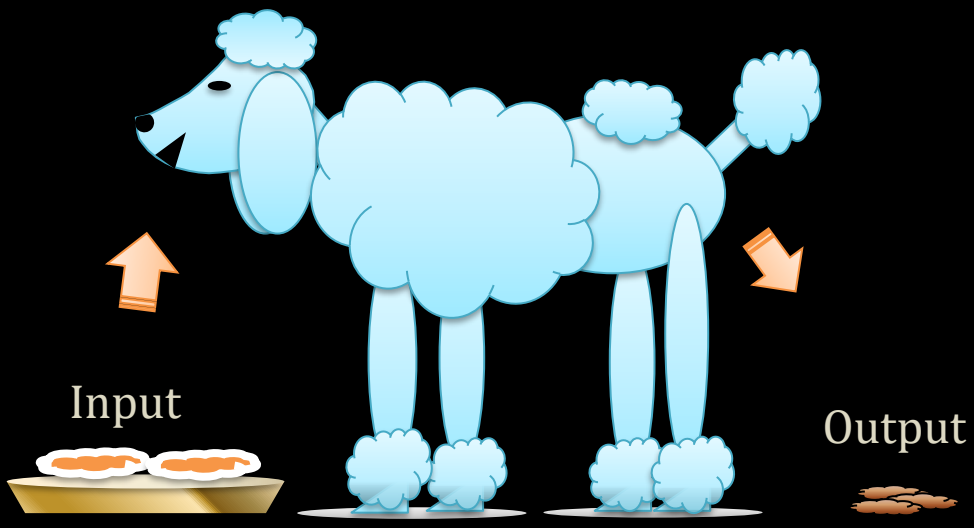
Reintroduction to Algebra with Intuition

Algebraic Foundation for

Calculus, Computer Science, Physics, and Engineering

THIRD EDITION



Ruchira Sasanka

Reintroduction to Algebra with Intuition

Algebraic Foundation for

Calculus, Computer Science, Physics, and Engineering

Third Edition

Ruchira Sasanka, Ph.D.

Version 3.0

CONTENTS

PREFACE VIII

SECTION I: FUNDAMENTALS 10

1 BUILDING A RELATIONSHIP: BROUGHT TO YOU BY FUNCTIONS 11
1.1 Modeling with Math 12
1.2 Relationship Guide: How to Build a Relationship (Tip: Use a Function) 13
1.3 Visual Model (Mental Model) For a Function 14
1.4 Meet the Everyday Functions You Hardly Think About 15
1.5 Function Notation 16
1.5.1 Lazy Mathematicians: Concise Notation 20
1.6 Function Definition vs. Function Evaluation (Function Call) 21
1.7 Taking Selfies: Visualizing Functions 23
1.8 The More the Merrier: Functions with More Than One Input 24
1.9 Operators Are People, err…, Functions, Too! 26
1.10 It's Complicated: Building Complex Relationships 27
1.10.1 Using Function Composition to Build Complex Models 27
1.10.2 Adding and Multiplying Functions As Composition 32
1.11 There's More Than One Way to Skin a Cat (or a Relationship) 33
1.11.2 Using a Common Parameter 34
1.11.3 Going Beyond Functions 36
1.12 The Story So Far 37

2 SOLVING RELATIONSHIP PROBLEMS 39
2.1 Whodunnit: Which Input Caused That Output? 39
2.1.2 Models for Solutions 42
2.2 Oops! Hit Undo! Hit Undo! Undo with the Inverse of a Function 42
2.2.1 When Functions Refuse to Invert 44
2.2.2 Finding Inverse Functions 45
2.2.3 Using Inverse Functions to Solve Equations 47
2.2.4 Graph of Inverse Functions 48
2.2.5 Inverse Operators 50
2.3 Putting it All Together: Solving Equations 51
2.3.1 Bad Root! You Produced Nothing! 51
2.3.2 Rooting for the Inverse 53
2.5 The Story So Far 55

3 NUMBERS: WHAT FUNCTIONS CONSUME AND PRODUCE 56
3.1 Invasive Numbers: How Numbers Breed More Numbers 56
3.1.1 Addition and Multiplication 57
3.1.2 First Signs of Trouble: Subtraction 57
3.1.3 More Trouble: Division 59
3.2 Unwholesome Whole Numbers 59
3.2.1 Poor Abstraction of Reality 60

3.3 Operations Revisited (with Negative Numbers) ... 61
3.3.1 Addition and Subtraction ... 61
3.3.2 Multiplication as Two Operations ... 62
3.4 Go Forth and Multiply, err…, Exponentiate ... 66
3.5 The Story So Far ... 69

4 COMMON RELATIONSHIPS (FUNCTION TOOLBOX) ... 70
4.1 Power Function Family: How to Grow Your Power ... 71
4.1.1 Linear Term ... 72
4.1.1.1 Reducing Non-Linear Models to Linear Models ... 76
4.1.2 Quadratic Term ... 78
4.1.3 Cubic Term ... 81
4.2 Polynomial Family: How to Accumulate Different Powers ... 82
4.2.1 Linear model (Linear Function) ... 83
4.2.1.1 Significance of the Absence of a Constant Term ... 85
4.2.1.2 Linear Models of Multiple Inputs (Linear Combinations) ... 86
4.2.2 Quadratic Model ... 87
4.2.3 Cubic Model ... 90
4.2.4 Generalization: Polynomial Functions ... 90
4.2.4.1 Polynomials as Linear Combinations ... 91
4.2.4.2 The Fundamental Theorem of Algebra ... 91
4.3 Exponential Family ... 93
4.3.1 THE (Natural) Exponential Function ... 98
4.4 The Inverse Functions of The Families We Met ... 99
4.4.1 Inverse of Power Functions ... 100
4.4.2 Inverse of Polynomials ... 102
4.4.2.1 Inverse of the Linear Model ... 103
4.4.2.2 Inverse of Polynomials ... 104
4.4.3 Inverse of Exponential Functions: Logarithmic Model ... 105
4.5 Reciprocal Functions of the Families We Met ... 109
4.5.1 Reciprocal of Power Functions ... 110
4.5.1.1 Reciprocal of The Linear Term ... 110
4.5.1.2 Inverse Square Model ... 112
4.5.2 Reciprocals of Polynomials ... 114
4.5.3 Reciprocal of Exponential Models (Exponential Decay) ... 115
4.6 Trigonometric Family ... 117
4.6.1 Periodic Functions ... 117
4.6.2 Communicating with Waves ... 123
4.6.3 Inverse and Reciprocal Models of Trigonometric Functions ... 128
4.7 Growing Faster than Exponential: The Factorial Function ... 129
4.8 Building Larger Models from Simpler Models ... 132
4.9 Summary of Function Families ... 135
4.10 The Story So Far ... 137

5 SERIES: FUNCTIONS UNLIMITED ... 138
5.1 A Polynomial as a Series ... 138
5.2 The Sky is the Limit: Power Series ... 142
5.3 Maclaurin and Taylor Series ... 144
5.4 Fourier Series ... 149

5.5 The Story So Far 150

SECTION II: BEYOND FUNDAMENTALS 151

6 FUNCTIONS THAT CAUSE (YOUR HEAD TO) SPIN 152
6.1 A "Rotated" Number 155
6.2 Functions with Complex ("Rotated") Input and Output 157
6.2.1 Basic Arithmetic Operators (Functions) 157
6.2.2 Complex Multiplication as Two Operations 159
6.3 Complex Roots of Polynomials 160
6.3.1 Why do They Come in Pairs? 162
6.4 Graph of a Complex Function 163
6.5 Complex Numbers as "Complete" Numbers 166
6.6 Non-real Exponents (Advanced Topic) 169
6.6.1 Exponentiation with an Imaginary Input 170
6.6.2 Exponentiation with a Complex Input 174
6.7 The Story So Far 176

7 FUNCTIONS IN 3D SPACE 178
7.1 Vectors: Representing Objects in Space 178
7.2 Where do Babies, err…, Vectors, Come From? 179
7.3 Vector Difference (Subtraction) 180
7.4 Vector Addition 181
7.5 Multiplication by a Scalar (Scaling) 182
7.6 Examples of Vector Addition and Subtraction 182
7.7 Vector Functions You Meet Every Day 184
7.8 Product Between Two Vectors 186
7.8.1 Dot Product (Scalar Product) 186
7.8.2 Cross Product (Vector Product) 190
7.8.3 Why is Vector Product Not Commutative? 194
7.8.4 2D Vectors vs. Complex Numbers 195
7.9 Linear Combinations of Vectors 195
7.10 Component Representation and Algebraic Vectors 197
7.10.1 Addition and Subtraction in Component Representation 198
7.10.2 Dot Product in Component Representation 198
7.10.2.1 Dot Product as a Linear Combination of Components 200
7.10.3 Cross Product in Component Representation 203
7.11 What Unites Algebraic and Geometric Vectors? 204
7.12 Modeling with Vectors (Scalar Fields and Vector Fields) 206
7.12.1 Scalar Fields 206
7.12.2 Vector Fields 209
7.12.3 Vector Valued Functions of a Parameter 213
7.13 Function Space 214
7.14 The Story So Far 216

8 MATRICES: EXTENDING LINEAR FUNCTIONS 218
8.1 Linear Combination Revisited 218

8.2 Multiple Linear Combinations 219
8.3 What Does a Matrix Represent? 224
8.4 Modeling with a Matrix 228
8.4.1 Extending Linear Models 230
8.5 Matrix Multiplication 230
8.6 Matrix Multiplication as Function Composition 232
8.7 Row View vs. Column View of a Matrix 235
8.8 Linear Transformations 237
8.8.1 Performing Multiple Transformations at Once 239
8.8.2 Multiplying by a Matrix as Two Operations in One 240
8.9 Solving Linear Models with Multiple Inputs 241
8.9.1 Solving a System of Linear Equations 242
8.10 The Story So Far 244

9 SUMMARY: FUNCTIONS IN PERSPECTIVE 245
9.1 The Meaning of Life, err…, Math 245
9.2 Objects 246
9.3 Functions 246
9.3.1 Linear Models 247
9.3.2 Non-Linear Models 250
9.4 The Story 252

EPILOGUE 255

SUPPLEMENT: PRELUDE TO CALCULUS 257

ACKNOWLEDGEMENTS 264

INDEX 265

ABOUT THE AUTHOR 267

PREFACE

This is the math book I wish I had when I was in high school and college. If you were like me, by the time you got to high school or college, you would have learned a considerable amount of math, mostly algebra. However, you may feel that what you have learned is a bunch of jumbled up ideas and a myriad of ways to manipulate numbers and symbols. To make matters worse, you may feel that you could hardly explain why you ever learned those things, or how one thing you learned is related to another. That's how I felt about math for a very long time. If you feel the same way, this book will give you an opportunity to take a fresh look at what you have learned, and re-learn previously learned concepts (and more) in a new light — one that will develop your intuition and help you see how everything is connected to serve a common purpose in the real world. That intuition will build a solid algebraic foundation to help you pursue other areas like Calculus, Computer Science, Physics, and Engineering.

This book attempts to present a narrative of the function dynasty: how they came to be, their purpose in life, what they have in common, what they consume and produce, their kith and kin, how they appear in various forms in disguise, and how everything we learn in algebra is related to them in one way or the other. If you are a student who is trying to make sense of what you are learning, or a parent or an educator looking for ways to explain math to kids in a cohesive and intuitive way, this book will give you many fresh ideas. It offers a lot of analogies, figures, visual models, graphs, real-world examples, and gratuitous explanations to help anyone connect with the fundamentals of math. Instead of formalism, this book places emphasis on developing intuition.

There is another unique feature of this book. It tries to explain the connection between math and computer programming with numerous examples of very short computer routines that correspond to their mathematical counterparts. These routines are written in JavaScript-*like* syntax to help anyone digest them easily. Computer science students and anyone who is even remotely interested in computer programming will find the connection between math and computer programming quite enlightening. You will soon realize that understanding one helps you better understand the other. Similarly, this book should help you better understand models used in many other scientific disciplines.

Finally, I should also mention what this book is not. This is not a textbook. This is not a test preparation guide. This is not a cheat sheet. This book was written to serve one purpose — to develop your intuition and help you see the underlying connections between different, seemingly unrelated math concepts. That understanding will allow you to appreciate math as a real-world tool, the same way engineers and scientists do. There is no guarantee that

this book will improve your test scores. On rare occasions, it might lower your scores. Other side effects may include nausea, dizziness, and allergic reactions to dry humor. If symptoms persist, you should immediately stop reading this book and read a standard text book.

Ruchira Sasanka

SECTION I: FUNDAMENTALS

1 Building A Relationship: Brought to You by Functions

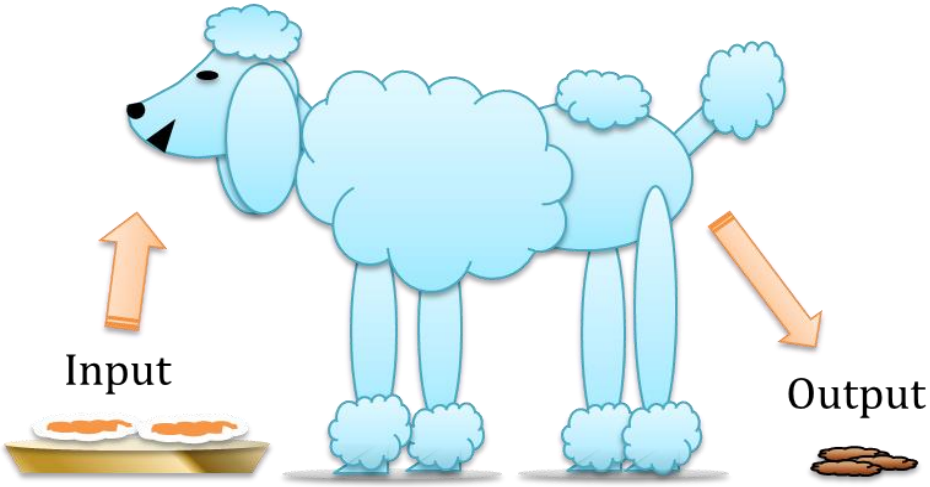
Chapter Overview: This chapter introduces you to mathematical relations and functions, the basic modeling tool in math, and describes the fundamental properties common to all of them. We will start with basics of function notation, function definition, evaluation, and visualization. Then we will explore how functions are related to operators and how we can use function composition to build more complex models using simpler models. Finally, we will see how a common parameter can be used to model more complicated relationships.

Why do we study math? What's the meaning of all this number crunching and formula juggling we learn in school? Although it may sound like a deep philosophical question that needs lots of soul searching, the answer is simpler than you think.

With math, we try to build various models about the world we live in.

"But", you may object, "that looks like a really difficult thing that requires a lot of expertise". Not so! It is much easier than you think.

Let's start small. Let's start with your dog.



Modeling your pooch with functions

Let's say you realize that if you feed your dog some amount of dogfood a day, she produces half that amount of poop (in weight) the next morning. For instance, if you give her 6 ounces[1] of dogfood, she poops 3 ounces (sometimes, don't you wonder whether she poops more than what she eats!)

[1] This book refers to units like ounces, pounds, and feet that are customarily used in the US. You can substitute them with any corresponding metric unit such as grams, kilograms, and meters as appropriate.

This is an example of a real-world relationship. The relationship is between the weight of dogfood your dog eats and the weight of poop she produces. You can simply model this as:

Weight of poop = Weight of dogfood / 2

Congratulations! You have built a "mathematical model", also known as a "**mathematical relation**". That is, a relationship between input and output.

A mathematical relation describes a relationship between input and output

Now you can test whether your mathematical model is actually true by observing the amount of dogfood (input) and poop (output) over many days. This is one use of a mathematical model. It allows us to **test** the validity of a theory (hypothesis) you have about the real world. In addition, from your mathematical model, you know that if you increase the amount of dogfood (input) by 1 ounce, the amount of poop (output) will increase by half an ounce. So, you can use your mathematical model to **predict** what would happen when you change the input. That's another use of a mathematical model.

1.1 Modeling with Math

The purpose of math is to develop mathematical models for various relationships that are important to us. A couple of such famous relationships, thanks to Newton, are:

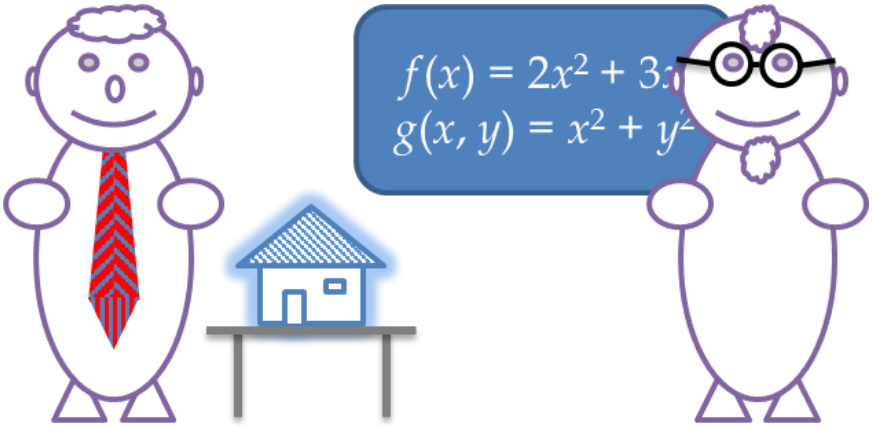
$$\text{Force} = \text{Mass} * \text{Acceleration}$$

$$\text{Gravitational Force Between 2 Objects} \propto \frac{\text{Mass of Object 1} * \text{Mass of Object 2}}{\text{Square of Distance Between 2 Objects}}$$

If you think your mathematical model about your dog was rather silly, consider this: you used the same math construct as Newton did! Of course, scientists can use Newton's mathematical models to send rockets to the Moon, but that's beside the point.

We use mathematical relationships to model the real world!

Math is the tool we use to understand the physical world around us. Just as an architect builds a miniature model of a building to understand the real one he is about to build, mathematicians, scientists, and engineers build "mathematical models" (or mathematical relationships) of the physical world.

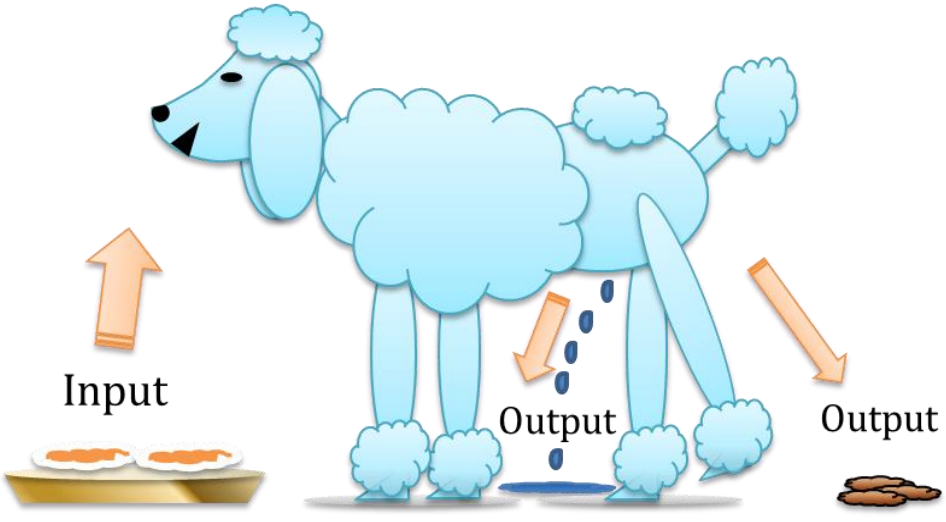


An Architects' Model vs. Mathematician's Model

1.2 Relationship Guide: How to Build a Relationship (Tip: Use a Function)

A relationship relates two (or more) things. When it comes to a mathematical relationship, the relationship is between "inputs" and "outputs" — i.e., how output is made from given input.

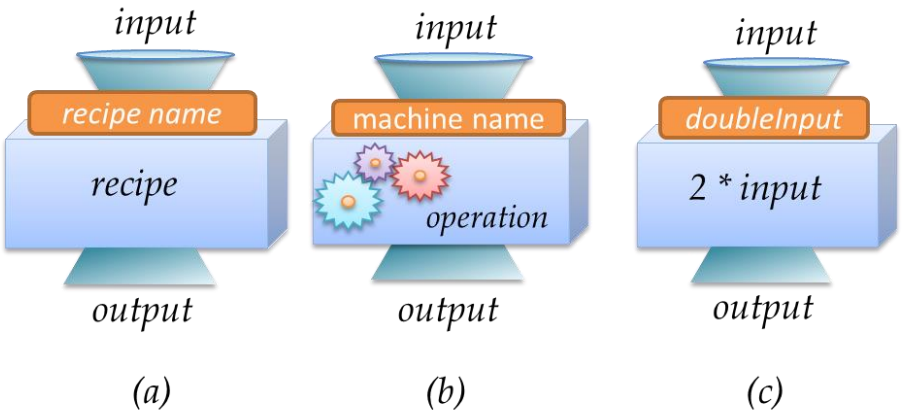
A mathematical relationship that produces only *one output* for a given input is called a mathematical function (or simply a **function**). In other words, a function cannot produce two outputs for the same input. If we want to model two outputs, we need two functions. Simple as that.



Bad function! No two outputs!

1.3 Visual Model (Mental Model) For a Function

It is quite helpful to have an intuitive visual model (or a mental model) of a function. The figure below shows such an intuitive **Visual Model**. In figure (a) below, we look at a function as a **recipe** that calculates output from input. Another intuitive way to look at a function is as a **machine**, as given in figure (b). The machine uses some operation to produce output from input. Figure (c) shows an actual example of a function, which doubles its input.



To summarize, a function describes a "recipe" or a "machine" for creating an output from given input(s) — i.e., the output is a "*function of*" the given input(s). Further, we usually give each function a distinct name, in order to identify it, as shown above. We will be using this Visual Model quite extensively in this book, so pay extra attention to it.

A function describes a "recipe" or a "machine" for creating an output from input(s)

As an example, figure (c) above shows a function (recipe) that doubles its input. The output is just twice, or double, the input. We have given this function a name — *doubleInput*.

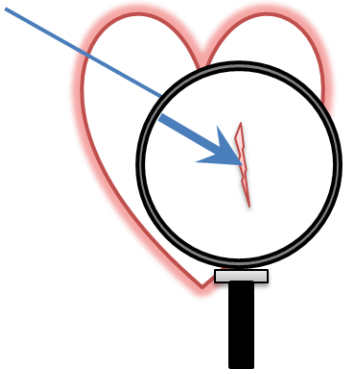
Some people like to think of a function as an "action" or an "operation". A function performs some "action" on the input to produce an output. Since we use a verb to describe an action, some find it more intuitive to use a verb in the function name to describe the action the function performs. For instance, we called the above function "doubleInput" with the verb "double" to indicate that this function performs the action of doubling the input it receives. This is not a hard and fast rule, but we will try to use it whenever it makes things clearer.

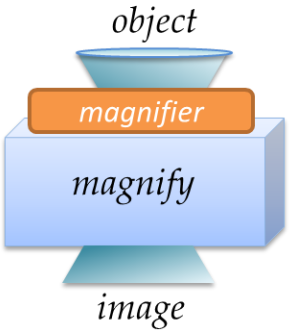
We will look at several example functions in the next section. For each example, take extra care to understand the Visual Model.

1.4 Meet the Everyday Functions You Hardly Think About

There are many everyday experiences that can be modeled as functions. These functions are hiding in plain sight because you hardly ever look at these as functions.

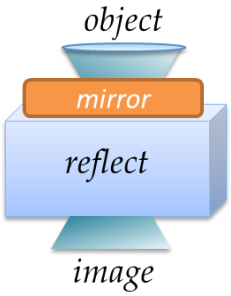
Think about a magnifier. It magnifies an object (input) and produces an output image (which you can see). Therefore, you can look at (or model) the magnifier as a function[2]. The figure below shows a magnifier and its Visual Model.





The above example will make you realize that you can think of your eye as a function. Similarly, do you see your eye-glasses and camera as functions? All of them use lenses to create an image (output) of an object (input). The eye produces the image (output) on the retina while the camera produces its image (output) on a film or an image sensor.

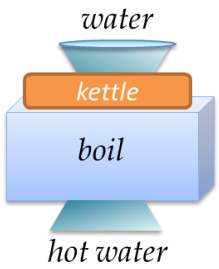
Even you can think of your bathroom mirror as a function. It creates a reflection or image (output) of the same size as objects in front of it (input), with one caveat; it switches left and right. Can you create a Visual Model for the mirror?



We discussed how we should look at a function as a recipe. The converse is also true. We can look at a recipe as a function. You take 1 pound of flour (input), ½ pound of sugar (input), 6 eggs (input), and 1 packet of yeast (input), and you can make a cake (output) of 1 pound. I hope. I didn't say it would be edible! Notice that this function has many inputs and produces one output (cake).

[2] More precisely we can model magnification (i.e., the size of resulting image as a function of the size of input object). For discussion in this section, we will ignore such subtleties to convey the larger point of being able to find functions everywhere.

Let's look at some machines that can be modeled as functions. How about the ice maker in your fridge? You put water in and get ice out. How about the electric kettle in your kitchen, or the water heater in your house? You put water in and get hot water out. Similarly, you put coffee nuts into your coffee grinder as input and get ground coffee as output. If you put strawberries, milk, and sugar as inputs into a blender, you can get strawberry milkshake as output. If you put potatoes into an oven as input, you can get baked potatoes as output for dinner. In each of these cases, the kettle, the water heater, the coffee grinder, the blender, and the oven can be modeled as a function producing output from input.

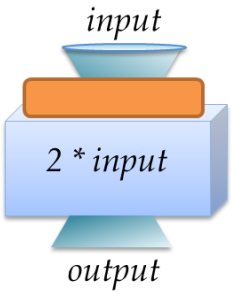


We encounter functions in our everyday lives!

I think you get the point now. Functions are not some exotic mathematical concept. They live among us all, but we hardly ever notice them. Poor functions!

1.5 Function Notation

The Visual Model of a function is an intuitive way to define a function, but we use text-based notation in math. If we want to specify a relationship, where the output is made by doubling the input, we can simply write:



$$\text{output} = 2 * \text{input}$$

or

$$y = 2 * x$$

where x is the input and y is the output.

Note that symbols like x and y are just names we use to represent input and output. They are also called **variables** since they are names for quantities that can vary. The input variable x is also called the *independent variable* because it can freely take any value and the output variable y is usually called the *dependent variable* because it depends on the input.

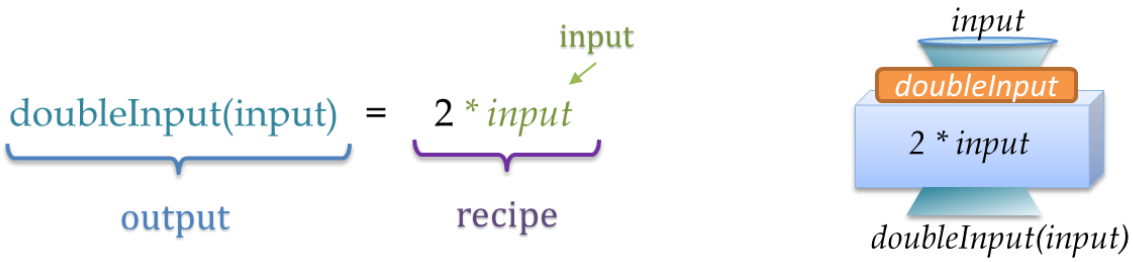
Consider the above relationship "output = 2 * input". This relationship is also called an **equation**, because it is expressed as an equality, using an equal sign. As its Visual Model shows, there is no "recipe name", or "function name" given to it; yet, it is a relationship between input and output. This works well for many modeling activities, but not always.

Imagine we had two relationships, one for doubling and one for tripling. Each time we want to refer to one, we don't want to say "the one that doubles input", and "the one that triples input", and so on. That is clumsy. Fortunately, this problem is quite easy to solve. If you had two dogs, you wouldn't call them "the one with the fluffy coat" and "the one with the pointy ears". You would just give them two names — Fluffy and Sparky. We do the same with functions, and that's why our Visual Model has a "name tag" called "recipe name" (function name). We can call a function that doubles its input "doubleInput". If we have a function that squares its input, we can call it "squareInput". That's pretty easy to understand.

Now that we have a name for the function, we have to define the recipe. Defining the recipe is called, you guessed it, "**function definition**". Let's see how we do this for our doubleInput function. It's really easy. We would write it as:

doubleInput (input) = 2 ∗ input

The following picture names each component of this input-output relationship, with its corresponding Visual Model on the right.



The above *function definition* has the same three things we have been talking about:

(1) input
(2) recipe
(3) output

First, let's look at the input. Input is represented by a variable; in this case, we have named it 'input'. If we want, we can call input anything we like — e.g., '*in*', '*x*', or '*t*'. It doesn't matter, as long as we give the input a name.

Next, we have the recipe. It's how we make output from the input. In this case, we just multiply input by two. Note that the recipe always goes on the right-hand side of the equal sign, in a function definition.

Last, we have the output, which appears on the left-hand side. Here, we have something new. Instead of having a simple variable name like "*y*", or "out", we have:

doubleInput (input)

This consists of:

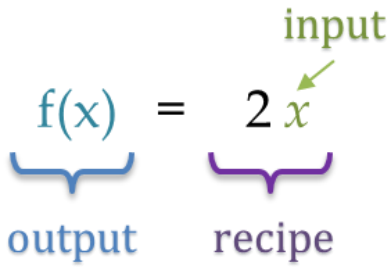
- Function name (i.e., doubleInput)
- Input variable within parenthesis (i.e., input)

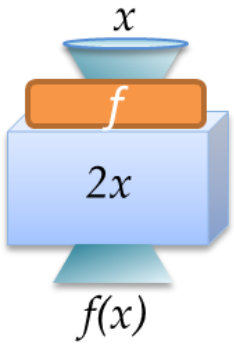
Why do we do this? Of course, a function needs a name, just as our dogs "Fluffy" and "Sparky" needed names. Now, rather than introducing a completely unrelated variable name like "y", what if we could come up with a name for the output using the function name itself? That's exactly what we do. We create a name for the output, using the name of the function and the name of the input variable. So, doubleInput(input) means the following: it's the output of the function named doubleInput, using the input named "*input*".

It is important to note that we can use any name for the input variable and to name the function. For example, we can write the above function definition as:

- doubleInput(in) = $2 * \text{in}$
- doubleInput(x) = $2 * x$
- $f(x) = 2x$

All of the above definitions describe the same recipe but use different names to do so. Carefully, study how the output is named in each of those definitions. Just to drive the point home one more time, you should look at the last definition $f(x) = 2x$, as an input-output relationship. In this case, the input is named x, the recipe is $2{*}x$, and the output is named $f(x)$ to indicate that it's the ***output of function*** f ***for input value*** x. Note that the output[3] term $f(x)$ is pronounced as "f of x". This definition and its Visual Model are given below.





If you find this convention of naming output of a function rather strange, here is an analogy that will make it clear. Think about the grater you have in your kitchen. If you were to grate cheese with it, you call the output, "grated cheese". If you were to grate carrots, you would

[3] Traditionally, notation $f(x)$ is used (or rather abused) both as the name of function and the output of function f at input x, based on the context. Here, we use the latter meaning for better intuition.

call the output "grated carrots". The output name is a combination of the action or function you do (grating) and the input (cheese or carrots). If you want another example, think about "grilled chicken", "grilled salmon", and "grilled veggies". Again, the output is a combination of the function "grilling" and the input (chicken, salmon, veggies, etc.). Similarly, we use $f(x)$ to name the output of the function f for input x. Who thought the output of functions could be named like fast food?

Before we move on, we should learn two words that are often useful in describing the recipe of a function definition. The recipe of a function (i.e., the right-hand side of function definition), is an **expression**. What's an expression? It's a sum of *terms*? OK, what's a **term**, then? A product or a ratio of constants and variables is called a term. For instance, $2x$, -5, $5x^2$, 61, 5^3, ½, and $x^3/2$, are all terms. Therefore, a sum of those terms, like $5x^2 - 2x + 6$, is an expression with 3 terms. You can look at term $2x$ as an expression with just 1 term.

If you want to look at function definitions from a different angle, looking at how we define functions in a typical programming language should fascinate you. The computer code for the function "doubleInput" is given below. It is strongly advised that you study this code, even if you are new to computer programming. You should immediately see the similarities between the function definition, our Visual Model, and computer code.

Computer code starts with the keyword "`function`" to say that this is a function definition. Then, it specifies the name of the function, "`doubleInput`", followed by the input variables ("`input`", in this case). Then, within curly braces, we have the body or recipe, which tells us how the output gets made from the input. You can see that the output is calculated by multiplying the input by two. Then, there is one additional "`return`" statement, which indicates that the function *outputs* (or "returns") the variable named "`output`". That's it!

```
function doubleInput( input )
{
      output = 2 * input
      return output
}
```

Depending on the specific programming language, you may need additional keywords to specify that "`input`" and "`output`" are variables. We will omit such syntax for clarity. Our

objective is to see the fundamental connections between math and programming, compared to learning the syntax of a specific programming language.

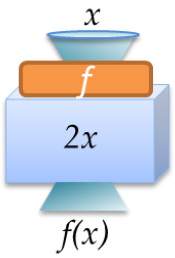
When defining a function, we said that the names of the input and output variables do not matter. This is also true for computer code. To illustrate this point, the following figure shows the same `doubleInput` function, but with the input variable named "`number`" and the output variable named "`result`". Carefully compare this with the previous version. The actual operation that the function performs did not change at all. Only the names of the input and output changed.

```
function doubleInput( number )
{
        result = 2 * number
        return result
}
```

This example helps you appreciate the connection between math and computer programming. This connection will become more evident as we go on.

1.5.1 Lazy Mathematicians: Concise Notation

In our previous examples, we defined our functions and input/output variables with descriptive names like "doubleInput", "*input*", and "*output*". However, mathematicians, who do not care much for writing long names, would like to give single letter names to functions and variables. For example, the function definition we wrote as:



$$\text{doubleInput (input)} = 2 * \text{input}$$

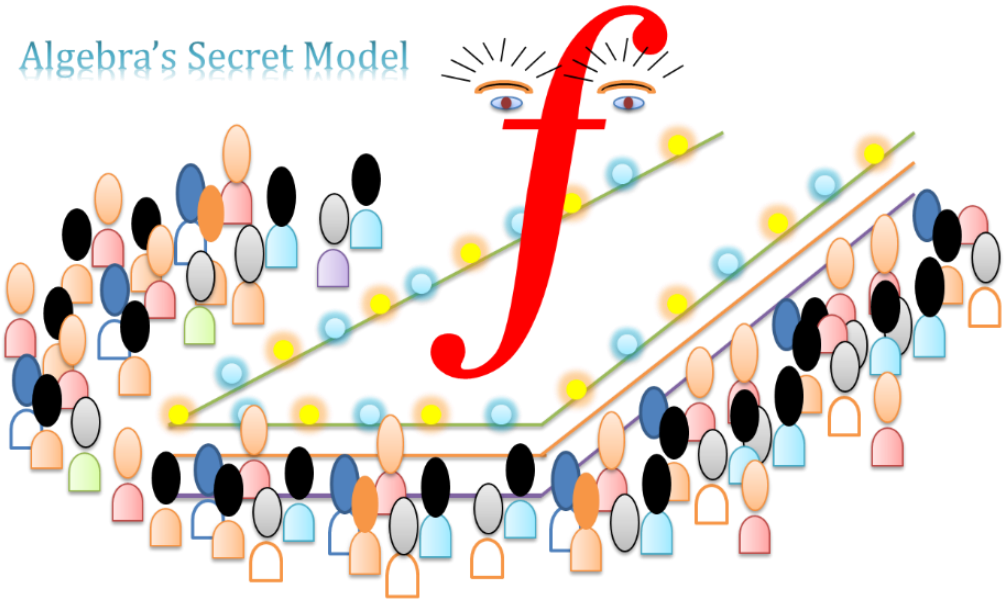
is often written as:

$$f(x) = 2x$$

As you can see, the recipe is the same. Only the names are shortened to single letters. Instead of naming the function "doubleInput", the letter "f" is used as the function name. Instead of using the variable name "input", the letter "x" is used.

In contrast, computer scientists advocate for using long, descriptive names. Short names could be challenging to many students who can't make much sense of the sea of letters they see on a page. However, those who text "cu @ 9 ttyl" to mean "see you at 9, talk to you later", don't have a right to complain.

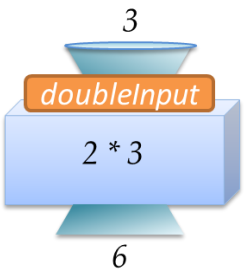
Kidding aside, we need to get accustomed to the mathematical notation. To help with this, in the first few chapters, this book will present both forms whenever possible.



1.6 Function Definition vs. Function Evaluation (Function Call)

In Section 1.5, we discussed the "definition" of a function. For instance, $f(x) = 2x$ is a function definition. Remember, a function definition has a left and right side, with the recipe on the right and output on the left, with an equal sign in the middle.

In contrast, the Visual Model on the right shows a function **evaluation** for a *specific* input value (i.e., in this case, the input is a constant). Here, the specific value, 3, is used as the input to the function and hence the function outputs 6 (i.e., 2∗3). The evaluation of the function (i.e., finding the specific output value for a specific input) is written as:



doubleInput(3) [evaluate function doubleInput with 3 as the input]

The computer code for this is given below. The text following // are just comments. The print statement prints number 6. Do you see the similarity between math and computer code?

```
y = doubleInput( 3 )      // call doubleInput with 3 as input
print( y )                // prints 6
```

Since a function evaluates to a single output value, it can be used as any other *term* to form expressions. For instance, the following expression results in 30, if function f is defined as $f(x) = 2x$, because $f(3)$ evaluates to 6 (i.e., $2*3$).

$f(3) * 5$ [this evaluates to 30, if $f(x) = 2x$]

$f(3)$ represents the output value of function f for the input value 3

The following figure shows the same expression evaluation with computer code. There, we multiply the value of doubleInput(3) by five. As we saw above, doubleInput(3) evaluates to value 6. Then, we multiply this result by 5 and assign it to the variable y. Therefore, the print statement prints the number 30.

```
y = doubleInput( 3 ) * 5
print( y )                      // prints 30
```

If you are still unsure about the difference between the function definition and evaluation, here is an analogy. Think of an electric kettle. It can be modeled by a function. You put cold water in (input) and after a while, you get boiling water (output). Some electrical engineer came up with instructions (recipe) for building that kettle. That's the definition. A factory could manufacture the kettle according to that recipe from the definition. Once the kettle is made, we don't deal with the definition anymore. A consumer buys the kettle and "uses" it — puts cold water in and gets boiling water out. That "use" of a function is called "evaluation".

It is very important to understand the difference between function definition and function evaluation. Function definition gives a **recipe** for producing an output for ***any*** given input (input and output are variables). Function evaluation produces one specific output for one ***specific*** input (input and output are constant).

As another example of function evaluation, if we plug in constant 'a' as the input to function 'f', the output value can be represented by $f(a)$, as shown in the Visual Model. This is analogous to evaluating $f(3)$.

$f(a)$ represents the output value of function f for input value a

This notation nicely jibes with the notation we used for function definition. We saw that if we use 3 as the input for f, then we can represent the output by $f(3)$. If we use constant 'a' as the input, the output can be represented by '$f(a)$'. What happens if we use variable 'x' as the input? We can represent the output by $f(x)$. That is, $f(x)$ is the output value that comes out of the function for input x. Rather than introducing another variable (e.g., y) to represent the output, we can use $f(x)$ itself to represent the output of function f, when input is x. This further clarifies why we used $f(x)$ to represent output in function definition.

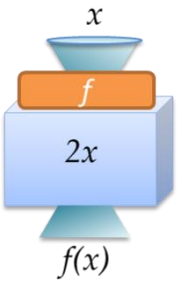
$f(x)$ represents the output value of function f for the input value x

1.7 Taking Selfies: Visualizing Functions

A picture is worth thousand words. Given that math expressions are really concise, a picture is worth a thousand math symbols when it comes to developing intuition.

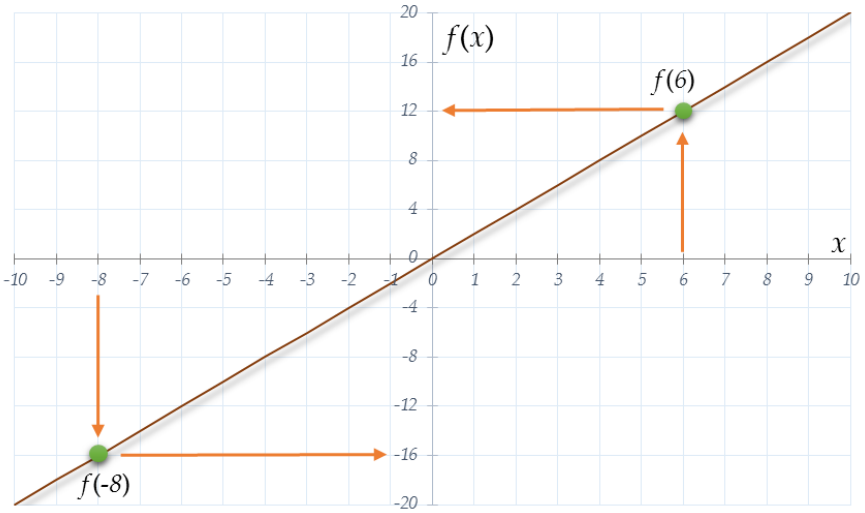
A function can be visualized by drawing its **graph** — i.e., for each input value, we plot the corresponding output value. However, it is important to know the difference between a function and its graph. Your selfie is not same as you. Similarly, a graph is a mere visualization of a function. A function represents a mathematical relationship between inputs and output.

The following figure shows the graph of function, $f(x) = 2x$, which is shown in the Visual Model. This function is similar to doubleInput, but here, we use the shorter name f and input variable x to save space on the graph.



The graph can be drawn by calculating the output for each input. For instance, for an input value of 6, the output is 12, as shown on the graph. Also, note that we can represent this output value with expression $f(6)$, as

we saw in Section 1.6. Recall that $f(6)$ indicates the function evaluation (output), when x is 6. Similarly, the output for input -8 is $f(-8)$, which is equal to -16 as shown on the graph.



Each point on the horizontal axis represents an input value, x, for the function. The values on the vertical axis shows $f(x)$, the output value, corresponding to an input value of x.

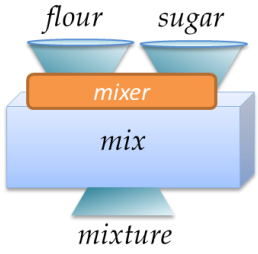
From the graph of $f(x) = 2x$, we can immediately see that the relationship between the output and input is linear (i.e., a line). That is, if we increase the input value by some increment, the output also increases by a constant multiple of that increment. For instance, $f(6)$ is 12. If we increase the input by 3 (to 9), the output increases by 2∗3 = 6 (to 18). You can verify that by evaluating $f(9)$, which is 18.

A graph of a function helps us visualize the relationship between input and output

For the above graph, the input variable x takes values from -10 to +10. The set of input values a function accepts is also known as the **domain** of the function. The output of the above graph takes values from -20 to +20. The set of output values a function produces (for a given set of inputs) is known as the **range** of the function.

1.8 The More the Merrier: Functions with More Than One Input

Many real-world models need more than just one input. Think about a cake recipe. It has multiple ingredients (inputs). The Visual Model to the right shows another example: a mixer that accepts flour and sugar to produce a mixture. If we input 2 pounds of flour and 1 pond of sugar, we get a mixture weighing 3 pounds. Here is

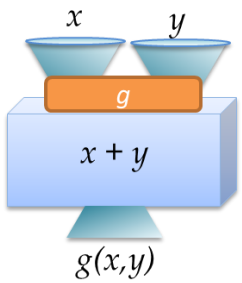


the function definition of this mixer, where all inputs and outputs are measured using the same unit (e.g., in pounds or kilograms):

mix(flour, sugar) = flour + sugar

Here is the same definition using more concise notation:

$g(x, y) = x + y$



The above function g takes in two inputs, x and y. The recipe is just to add them together. The notation $g(x, y)$ says that g is a function that accepts two inputs, x and y. Expression $g(x, y)$ represents the output, the same way $f(x)$ represents the output of function f with only one input, x. The above Visual Model captures this definition.

The same function with more descriptive names can be defined as:

addTwoInputs (input1, input2) = input1 + input2.

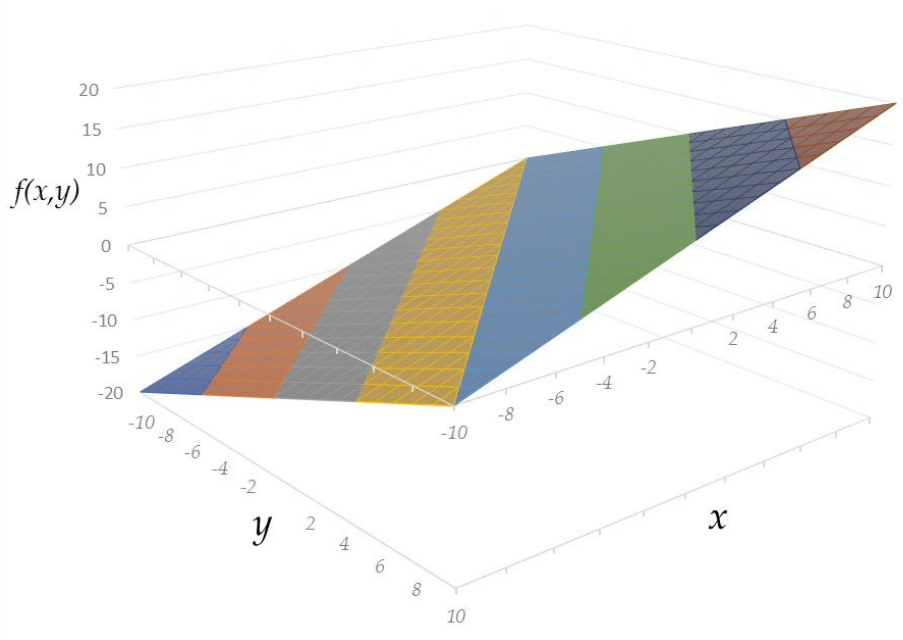
The computer code for this function, with more descriptive names, is given below, followed by a call to the function (evaluation).

```
function addTwoInputs( input1, input2 )
{
      output = input1 + input2
      return output
}
```

```
y = addTwoInputs( 5,3 )      // add 5 and 3
print( y )                   // prints 8
```

Here is a quick quiz. What does doubleInput(5) + addTwoInputs(3, 2) evaluate to?

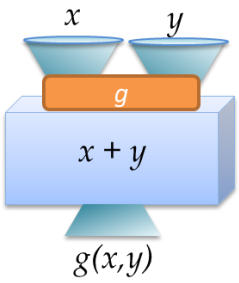
The figure below shows the graph of $f(x) = x + y$, which is same as function addTwoInputs. As you can see, the graph is a surface (a plane in this case). This is because we need two axes to represent both inputs x and y, and a third axis to represent the output value at each (x, y) point. The output value of function f at point (x, y) is represented as $f(x, y)$.



As the number of inputs increase, it becomes harder to visualize functions. We will see such examples in Section 7.12.1. As an exercise, can you draw a Visual Model to represent the magnification of a magnifier? Hint: Magnification depends on (i) focal length of the magnifier, and (ii) the distance between the magnifier and the object. Can you write computer code for your model?

You may notice that many algebra courses and textbooks treat functions with multiple inputs as an advanced topic. However, even a simple real-world model like $F = ma$ is a function of two inputs. For the same reason, most functions you write with computer code have multiple inputs. Therefore, we treat functions with multiple inputs as fundamental. In fact, the next section is a prime example of functions with multiple inputs.

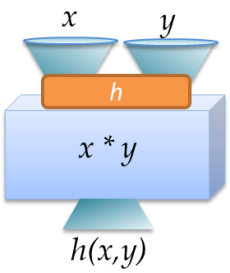
1.9 Operators Are People, err…, Functions, Too!



You may never have thought of arithmetic operators like addition, subtraction, multiplication as functions. But they definitely fit the bill as indicated by function addTwoInputs and function g shown with the Visual Model.

Function g and function addTwoInputs represent the addition operation. They take two inputs and produce an output, which is the sum of two inputs. Similarly, we can define the multiplication operation with another function as:

$$h(x, y) = x * y$$



or with more readable names as:

multiplyTwoInputs (number1, number2) = number1 * number2

The computer code for multiplyTwoInputs is given below:

```
function multiplyTwoInputs( number1, number2 )
{
      output = number1 * number2
      return output
}
```

As you can see, `multiplyTwoInputs(5, 3)` should evaluate to 15. We get the same result for $h(5, 3)$.

Can you draw the Visual Models and write computer code for the subtraction and division operators? Can you do the same for the negation operation (i.e., multiply by -1), which has only one input ?

Operators are functions too!

If you happened to treat operators as some strange things, now you know better. They are functions too. Isn't it nice to have one unifying concept that can unify all of these seemingly unrelated concepts?

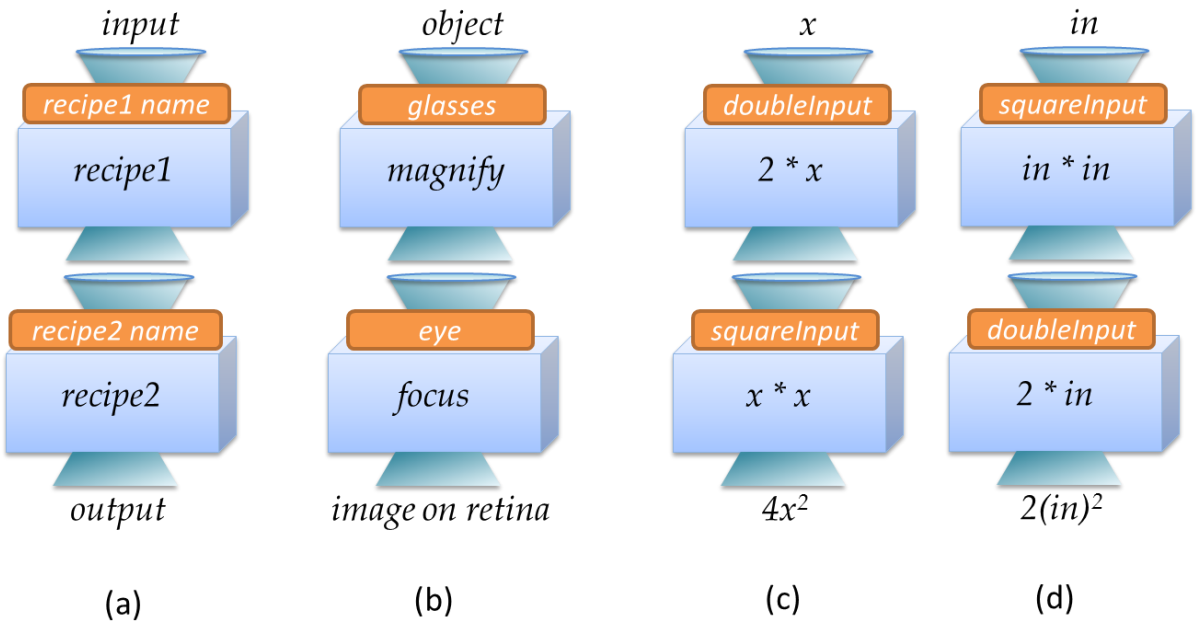
1.10 It's Complicated: Building Complex Relationships

Complex things are often built by combining simpler things. You cannot build a house with just one material; you need to combine multiple materials and objects. Similarly, a simple function cannot represent complex real-world relationships. Don't worry! There is a solution. We can combine simple functions to build elaborate models. This is called function composition.

1.10.1 Using Function Composition to Build Complex Models

Let's look at a few real-world examples. We discussed how the lens in your eye can be modeled as a function. It produces an image (output) of the objects (input) in front of you on the retina. But sometimes, this function is not sufficient. This is what happens when you develop near-sightedness or far-sightedness. Your eye cannot create a focused image of

objects that are too far away or too close. So, what do you do? You wear eye-glasses or contact lenses. We saw that glasses (or contacts) can be represented as yet another function, because they also produce an image (output) from input objects. What are we doing in this case? We are feeding the output of one function (glasses) into the input of the other (eye). In other words, we are combining (composing) two functions together to build a more complex function.



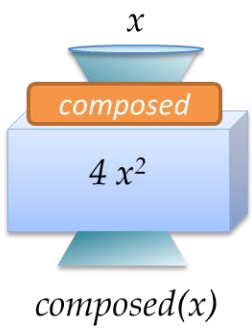
The above figure illustrates the general concept of function **composition**. As you can see, function composition can be viewed as feeding in the output of one function (top function) into the input of another function (bottom function). Figure (a) shows the general case of function composition. Figure (b) illustrates the eye-glass example, where the output of eye-glasses is being fed into the input of the eye.

We can compose more complex functions using simpler functions

Figure (c) shows an algebraic example, where function doubleInput is feeding in its output into function squareInput. These functions are defined as:

doubleInput(x) = $2x$ [multiply input by 2]
squareInput(x) = $x * x$ [multiply input by itself]

Figure (c) shows that if we feed in x as the input to top function, we get a new function, $4x^2$, out of the bottom function. Why? When we feed in x to doubleInput, the top function, its output is $2x$. This becomes the input to squareInput, which squares $2x$, giving $4x^2$ as the output of the combined function. Therefore, the recipe for the combined function, named 'composed', is given by



composed(x)

composed(x) = $4x^2$

You can quickly verify this with a sample input. If input x is 3, doubleInput outputs 6; with that as input, squareInput produces 36. Equivalently, composed(3) produces 4*3*3, which is 36.

Algebraically, you can find the composed function using a similar process. To find the output of the combined function, composed(x), we have to feed in doubleInput(x) as the input to squareInput(x):

composed(x) = squareInput(doubleInput(x)) **(1)**

= squareInput ($2x$) [$2x$ is the output of doubleInput(x)]

= $2x * 2x$ [squareInput multiplies input by itself]

= $4x^2$

Expression doubleInput(x) represents the output of doubleInput, when x is the input. If the input is x, instead of doubleInput(x) we can write $2x$. Then we feed that $2x$ into squareInput producing $4x^2$.

Take another look at the Visual Model in Figure (c) and compare it with the algebraic manipulation. Both achieve the same result. One very important thing to notice is that we can represent the composition of two functions with one function (called composed(x)). As a result, we created a more complex function using two simpler functions.

To get an even better understanding of this, let's look at computer code. The definitions for doubleInput and squareInput should be really clear to you by now. The only new function is the composed function. There, to calculate the output, first you call (evaluate) doubleInput function and use its output as the input to the squareInput function. This is exactly what we did algebraically in equation (1) above.

```
function doubleInput( number1 )
{
      output = 2 * number1
      return output
}
function squareInput( number1 )
{
      output = number1 * number1
      return output
}
function composed( number1 )
{
      output = squareInput( doubleInput( number1 ) )
      return output
}
```

If it is still difficult for you to wrap your head around equation (1), we can write the same equation in two steps as given below. There, we first call the doubleInput function with `number1` as the input and assign the output to variable `output1`. Then, we use `output1` as the input to the `squareInput` function. Now, can you see that both versions achieve the same end result?

```
function composed( number1 )
{
      output1 = doubleInput( number1 )
      output = squareInput( output1 )
      return output
}
```

The algebraic formulation equivalent to the new composed function in computer code is:

$$\begin{aligned} y &= \text{doubleInput}(x) \\ \text{composed}(x) &= \text{squareInput}(y) \end{aligned}$$

Can you clearly see the similarity between the computer code and the algebraic form? Being able to see both forms leads to a better understanding. This is the fundamental reason that this book provides both forms. Moreover, it shows you the inherent connection between math and computer programming.

Function composition is feeding the output of one function into another's input

In figure(c), doubleInput feeds into the squareInput function. What if we change the order? What if squareInput was at top, feeding into doubleInput? That case is shown in figure (d) above. Do the two composed functions in figure (c) and figure (d) produce the same output? No! If we plug in x into composed function in figure (d), we get the output $2x^2$, not $4x^2$ (I have shown (d) with input 'in' to show you that the name of the input does not matter. What matters is the recipe).

This illustrates a property of function composition with far-reaching consequences. Mathematically, we say "function composition is not commutative". What does that mean in plain English? If you take scalar multiplication as a function, then $3*5$ is the same as $5*3$. That is, the order of inputs doesn't matter for scalar multiplication. We can switch input1 and input2 of multiplyTwoInputs function (multiplication operator). However, we cannot do the same with function composition. For instance, squareInput(doubleInput(x)) is **not** the same as doubleInput(squareInput(x)). In general, f(g (x)) is not equal to g(f (x)), where f and g are any two functions.

If you think about it a little bit using a real-world example, you will clearly see why this is the case. In figure (b) we used eye-glasses as a function. Similarly, we know that binoculars are a function (they magnify input). Say you are wearing glasses and looking through binoculars. What happens if you switch the order of the eye glasses and binoculars, so that the binoculars are closer to your eye and glasses are in front of the binoculars? You will not see anything! Why? Because in function composition, order matters. More technically, function composition is non-commutative.

In function composition, order matters (function composition is non-commutative)

You can carry out function composition repeatedly. For instance, when you are wearing eye glasses, and looking through binoculars, with the aid of your eyes, there are three functions involved — the binoculars, your glasses, and your eyes. The image that falls on your retina is a composition of those three functions — binoculars, glasses, and the lens of the eye, in that order. Algebraically, we can do the same.

You may be wondering why we are so obsessed with function composition. As we saw with eye-glasses example, function composition is useful to build more complex models using simpler models. Remember, the whole point of math is to model the real world. Combining simpler models to build more complex models is a necessary and invaluable tool. Function composition combines existing recipes (functions) to create new recipes.

Function composition is the tool for creating new functions from existing functions

Function composition like $g(f(x))$ shows that functions can also accept other functions as inputs and produce new functions as output. How do functions do that? Accepting the function f as input is similar to accepting the whole range of values produced by f. For each of those values produced by f, function g can produce a new output.

To summarize, functions can accept two types of objects as input and can produce two types of outputs as objects. The two types of inputs are:

(1) Numbers/Variables	(produces a **number/variable** as output)
(2) Functions	(produces a new **function** as output)

When a function accepts another function as input, it produces a new function as output and this process is called **function composition**. Thus, we use function composition to create new functions.

Functions can accept functions as inputs and produce functions as outputs

Function composition gives rise to its reverse operation, **function decomposition**, which is equally valuable. This allows us to break down a more complex operation into simpler operations, which helps both understanding and computation. Function composition and decomposition are heavily used in computer programming for this very reason. In programming, large functions are often decomposed into smaller functions. Then, we call one function, get its output, and use that as the input to call another function (as you saw in the computer code for the function "`composed`"). Programmers do this every day without even thinking much about it.

Function decomposition is breaking down complex functions into simpler functions

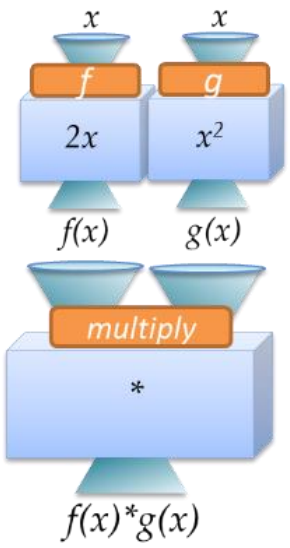
1.10.2 Adding and Multiplying Functions As Composition

Sometimes, function composition shows up in disguise. You may already know that you can create new functions by adding, subtracting, multiplying, and dividing two or more functions. For instance, by adding two functions, we can get one that quadruples the input:

$$\begin{aligned} \text{quadrupleInput}(x) &= \text{doubleInput}(x) + \text{doubleInput}(x) \\ &= 2x \qquad\qquad + 2x \\ &= 4x \end{aligned}$$

Similarly, by multiplying two functions, we can get one that cubes and doubles the input:

$$\begin{aligned} \text{cubeAndDouble}(x) &= \text{doubleInput}(x) * \text{squareInput}(x) \\ &= 2x \quad * x^2 \\ &= 2x^3 \end{aligned}$$



As you can see, this is an easy way to create more complex functions using simpler functions and therefore is essential in many modeling activities, as we will see later. However, here is something that is not completely obvious. Function addition and multiplication is a form of function composition. Why? Do you remember how we talked about operators like addition and multiplication as functions? So, what we are doing is feeding our functions into an operator, which is a function by itself, as shown by the Visual Model. This should not be a surprise because, to create a new function using existing functions, we need to use function composition.

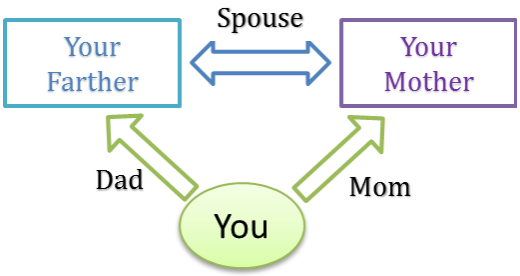
Note that this is different from feeding function f into function g. Instead, we feed both f and g into an operator, which is a function that takes two inputs. The operators can accept whole functions as inputs and produce new functions as their output. This shows that operators are more powerful and versatile than we thought before. In addition to accepting numbers and variables as input, operators can also accept other functions as inputs and produce new functions as their output. Being functions, operators get that ability naturally. In advanced mathematics, you will see this again with other operators like the differential and integral operator. Those operators also accept functions as inputs and produce new functions as their output. All of these are made possible by function composition.

If you didn't pay much respect to function composition, I hope I changed your mind. Among important concepts in algebra, it is second to only the concept of a function itself.

1.11 There's More Than One Way to Skin a Cat (or a Relationship)

In previous sections, we learned how to express a relationship between inputs and an output. However, it may not have occurred to you that there is more than one way to model a given relationship.

1.11.2 Using a Common Parameter



Let's take your parents as an example. They are related as spouses. Can they be related in another way? Let's see. You are related to both of them. So, we can establish another, and a different, relationship between your parents using you as the common thread. For instance, if you were to introduce your parents to another person, you can introduce them as husband and wife, or alternatively, as your mom and dad. Both are equally valid, though different relationships.

To make the above example quantitative, let's talk about the age of your parents. You may object to it, saying that it's not polite to talk about age of your parents. Well, you are asking too much from a book that begins by referring to dog poop. Anyways, I will not use their exact age, just because you asked nicely.

Let's just say your dad is 3 years older than your mom. We can express this relationship as:

$$\text{Dad's age} = \text{Mom's age} + 3 \qquad \textbf{(1)}$$

The above is a perfectly valid relationship. However, this is not the only way to express this relationship. Can you think of another way? I already gave you a big clue. Yes, it's you. Let's say your dad is 30 years older than *you*. Then you can express the age of your parents as follows:

$$\text{Dad's age} = \text{Your age} + 30 \qquad \textbf{(2)}$$
$$\text{Mom's age} = \text{Your age} + 27 \qquad \textbf{(3)}$$

We established a relationship between your dad's age and your mom's age using two relationships. These two relationships indirectly convey the same fact that your father is 3 years older than your mother, but that is expressed as two relationships, instead of one. Both relationships (2) and (3) use "Your age" as the input. Therefore, we call your age a common **parameter**. Using this common parameter, we can discover the original relationship given in (1).

You can do the same thing for mathematical functions. Sometimes, it is useful to look at a function using different relationships. Usually, we do that to make calculations easier or relationships more straightforward. For instance, the following example shows how we can reduce a 3-input function to a single-input function.

Let's say you want to bake cookies. To bake cookies, say you need four ingredients — flour, eggs, butter, and sugar. However, say that the final number of cookies is determined by the

weight of flour, butter, and sugar you use. You can model this relationship as a function of 3 inputs.

$$\text{Number of Cookies} = 10 * (\text{Weight of Sugar} + \text{Weight of Flour} + \text{Weight of Butter})$$

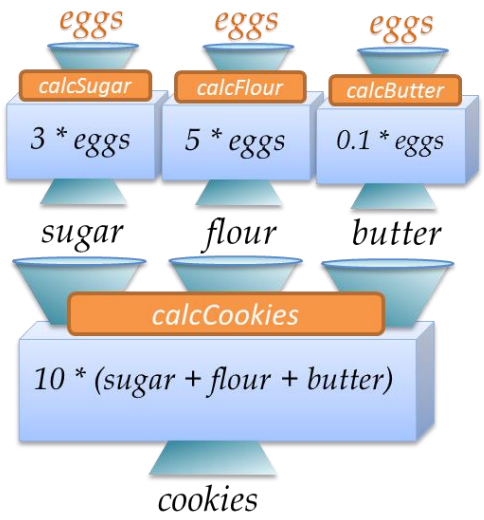
However, if you have even touched a mixer, you know it doesn't work this way. You can't take any amount of sugar, any amount of flour, and any amount of butter and make cookies. You have to take them in proper proportions. What does that mean in the context of functions? The three ingredients we have as inputs to the function are not really independent; they are related. You cannot independently assign any value you like to each. Rather, we have to pick one ingredient, and based on that, pick the amounts of other ingredients. Let's say, we use the number of eggs as the primary ingredient (independent variable), mainly because, we need to pick a whole number of eggs. Then, we can decide the weight of other ingredients based on the number of eggs we decide to use.

$$\text{Weight of Sugar} = 3 * \text{Number of Eggs}$$

$$\text{Weight of Flour} = 5 * \text{Number of Eggs}$$

$$\text{Weight of Butter} = 0.1 * \text{Number of Eggs}$$

Let's assume that all weights are in pounds, but you can use any unit like kilograms, or even metric tons, since we are not actually going to make any cookies. If you are reading this book for cooking advice, you have bigger problems you need to solve first!



What did we just do here? Since the inputs were related, we picked one variable as the primary input variable. Again, we call such a variable a *parameter*. Then, we can express all other variables using this common parameter. The above Visual Model shows this clearly. In our case, we used the number of eggs as our parameter. We calculated the weights of

sugar, flour, and butter, using this parameter. Using these three weights as inputs, we can calculate the number of cookies, with the help of our original function. Better yet, we can calculate the number of cookies using our common parameter as the sole input:

$$\begin{aligned}\text{Number of Cookies} &= 10 * (3 + 5 + 0.1) * \text{Number of Eggs} \\ &= 81 * \text{Number of Eggs}\end{aligned}$$

Therefore, our function of 3 variables became a function of just one variable, with the choice of a suitable parameter. With this parameter, we can express both the inputs and the output of the function.

You could have parameterized this relationship using a different parameter — e.g., you could have picked the weight of flour. The best parameter depends on the situation. In our example, if we picked the weight of flour as our parameter, and then chose some arbitrary amount of flour, we would end up with a fractional number of eggs, which would have been really messy when it came to baking cookies.

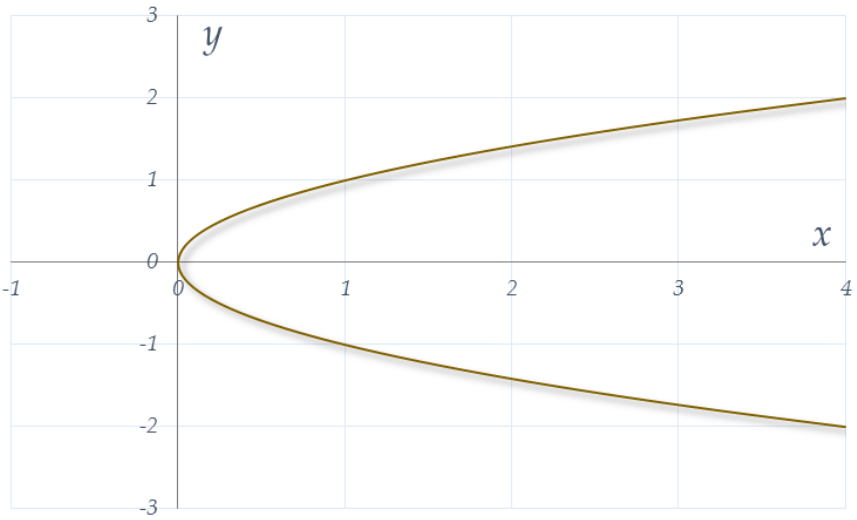
1.11.3 Going Beyond Functions

In many real-world problems, you will encounter models with a parameter. We can parameterize any relationship, but we usually do it when it is useful. In physical relationships, *time* is such an often-used parameter, since many other properties depend on time. For instance, if an object is moving through space, its vertical position (y) and horizontal position (x) vary as time goes on, and hence can be modeled using time (t) as a parameter, as follows:

$x = 4t^2$ [horizontal distance varies quadratically with time t]

$y = 2t$ [vertical distance varies linearly with time t]

As you can see, the relationship between x and y is expressed using time, t, as the common parameter. If we graph the relationship between x and y, we get the following graph. Note that the relationship between x and y cannot be expressed as a function of x, because, you have two output (y) values for the same input (x) value. For instance, when x=1, output y can take two values, 1 and -1. However, parameterization allows us to express this relationship between x and y as two proper functions of t. This is another advantage of parameterization.



Further, if we wanted to express the distance to a point (x, y) from origin, using Pythagorean Theorem, we get:

$$\begin{aligned} \text{distance} &= (x^2 + y^2)^{½} \\ &= (16t^4 + 4t^2)^{½} \end{aligned}$$

You can see that we can model distance as a function of one variable, instead of using two. This is another advantage of parameterization.

The above relationship models a parabolic curve. You can do the same for other curves like circles, ellipses, hyperbolas, etc. and often parametric equations are used in modeling such shapes that cannot be modeled with a single function. Expressing a complex relationship as a set of parametric functions makes analysis easier. As a result, parametric functions are a useful tool in modeling and analysis, and you will see another example in Section 7.12.3.

1.12 The Story So Far

This book is a story about the function dynasty. In this very first chapter, we looked at how functions came into existence in the first place, what they look like, and what traits they have in common.

Functions arose because of our desire to model the real world. We saw that all members of this function dynasty do just one thing: they produce output from input. That's all they do, but they do it really well.

As members of any family, functions can be quite diverse. Yet, they have a lot of things in common. We dedicated this chapter to those features that are common to all members of the family.

First, let's start with the simplest thing they share in common — a name. Whereas ordinary people get names like Robert and Chris, functions, being members of a dynasty, get fancy names, like f, g, and h. Some functions, called operators, get still fancier names like +, −, ∗, ÷, etc. They are called operators, because they are quite popular and useful in building many common relationships. The name of a function can be combined with its input to name the output. This works really well, because a function produces only one output, although it can accept one or more inputs.

Second, we saw that functions, just like celebrities, are photogenic. If we plot the output of a function vs. its input, we get a graph (a 'portrait') of a function, which is quite useful in identifying how the output varies when we change input (i.e., the shape of the function).

Third, we discovered a behavior of functions that makes them really successful as a family: they can cooperate with each other to get big jobs done. Put simply, a function can feed its output into the input of another. This has a fancy name: "composition". What it does is pretty mundane, yet astonishingly powerful.

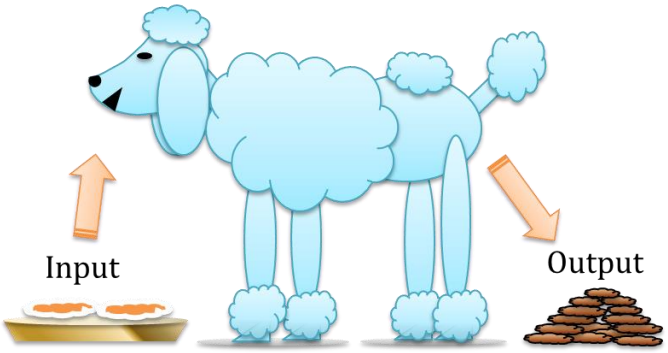
Fourth, in a big family, you can always find multiple relationships between given two persons. Your grandma is the mother of your mother, wife of your grandpa, mother of your uncle, and neighbor of your nephew Greg, and so on. Different relationships help us build different models, some of which are easier to analyze and some of which cannot be even represented by a single function. We saw that with parametric models.

Although functions are really powerful, as we all know, powerful people also have their own problems. In the next chapter, we are going to take a sneak peak at their messy problems.

2 Solving Relationship Problems

Chapter Overview: In the previous chapter we looked at how to build mathematical relationships using functions and explored traits common to all of them. Although we discussed how to evaluate function output when input is given, we did not explore the reverse — how to find the input that produced a given output. This chapter is dedicated to that topic. In the process we will meet inverse functions that "undo" operations, roots that lead to zero outputs, and finally, how these concepts can be used to find solutions.

The first chapter of this book looked at how to build a relationship. In real life, relationships always lead to messy problems! Mathematical relationships are no exception. Luckily in math, we have some tools to deal with those pesky little problems.



Gosh! What caused this much output?

Remember our pooch from the previous chapter? There, we built a model saying that she would poop half of dogfood she ate, as measured by weight. As an example, this model told us that if we gave her 200 grams of dogfood, she would produce 100 grams of poop. However, there is another very interesting question this model can answer. One day, all of a sudden if you got 500 grams of poop, you would no doubt wonder how much she ate (and who left the fridge open). This is a very common question we ask from any model we build. Although it is easy to answer in the case of the model we built for your pooch, it is always not so easy for the real-world models we build. That's why we have this entire chapter to build some background to help us answer that important question.

2.1 Whodunnit: Which Input Caused That Output?

As we saw in the previous chapter, finding the output with a given input is known as function **evaluation**. However, there is another important question we can ask: if we got

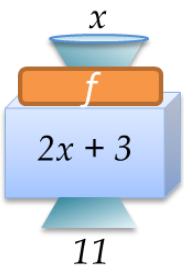
this output from a function, what was the *input* to the function that produced that output? This is known as **finding solutions** (or solving for input). For instance, take the function

$f(x) = 2x + 3$ [multiply input by two and add three]

If we have input 4, the output of the above function is given by:

$f(4) = 2*4 + 3$
$= 11$ [$f(4)$ evaluates to 11]

The above is function evaluation. However, let's say we want to find *which input* value produces the output value 11, as shown in the Visual Model to the right. Then, we have:



$f(x) = 11$ [function produces 11 as output]
$2x + 3 = 11$ [use recipe of function on the left]

We find the input value that produces 11 by *finding solutions* to the above equation, as:

$2x + 3 = 11$ [recipe produces output 11] **(1)**
$2x = 11 - 3$ [subtract 3 from both sides]
$x = (11 - 3) / 2$ [divide each side by 2]
$x = 4$ [Eureka! the input is 4]

Eureka! We found the input value that produced the given output. The above process is sometimes referred to as "solving for x".

It is very important to understand the difference between above two processes. One is function evaluation. We use that when we *know the input* and want to find the output. The other is finding solutions (to an equation). We use that when we *know the output* and want to find the input.

We need to solve equations to find input, when output is given

What are practical use cases of each? Say you are selling candy bars online. You build a model to find the price to charge a customer. You charge $3 for each candy bar and $1 for shipping the package, giving you the following model, where the input 'bars' represent the number of candy bars:

find_price (bars) = 3 * bars + 1

The Visual Model for it is given on the right. The computer code for this model is also straightforward:

```
function find_price( bars)
{
      price = 3 * bars + 1
      return price
}
```

How do we use this model? A customer may come and ask for the price of 10 candy bars. You use your model, with *input* of 10, to get the output price of 31 dollars. Another customer may come and ask you how many candy bars she can buy for 34 dollars. For that, you use 34 as the *output*, and solve for the input as:

bars ∗ 3 + 1	= 34	[3 bars plush shipping is $34]	**(2)**
bars * 3	= 34 – 1	[subtract 1 from both sides]	
bars	= (34 – 1)/3	[divide both sides by 3]	
bars	= 11		

Notice that when we know the output value and want to find the input, we always have an equation like (1) or (2). In general, this equation has the form:

$f(x) = b$ [output value is b for input x]

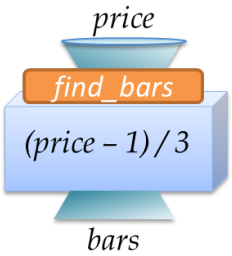
where b is the output value. For instance, in (2), b is 34 and $f(x)$ is $3x$+1. Therefore, whenever we have a function producing a specific output value, we have an equation of the above form.

A function producing output b yields to an equation of the form $f(x) = b$

You should understand that function evaluation and solving equations are two sides of the same coin. Real-world situations require us to use both of these methods under different circumstances. However, both of them involve a function, an input and an output. In one case, we know the input and want to find the output. In the other case, we know the output and want to find the input.

2.1.2 Models for Solutions

We can develop models for finding solutions too. For instance, we can build a more general model of (2), to find the number of candy bars for a given price:



bars ∗ 3 + 1	= price	[original model]
bars ∗ 3	= price - 1	[subtract 1 from both sides]
bars	= (price – 1)/3	[divide both sides by 3]

The Visual Model for the above solution is given on the right. Here is the computer code for it. It returns the number of candy bars when the price is given as an input.

```
function find_bars( price )
{
      bars = ( price - 1 ) / 3
      return bars
}
```

To summarize, we built two models:

find_price (bars)	= 3 * bars + 1	[find price when # bars is given]
find_bars (price)	= (price - 1) / 3	[find # bars when price is given]

Do you feel that these two models are somehow related? Of course, they are, because we could go from one to the other with simple algebraic manipulations as we saw above. We will find out their exact relationship in the next section.

2.2 Oops! Hit Undo! Hit Undo! Undo with the Inverse of a Function

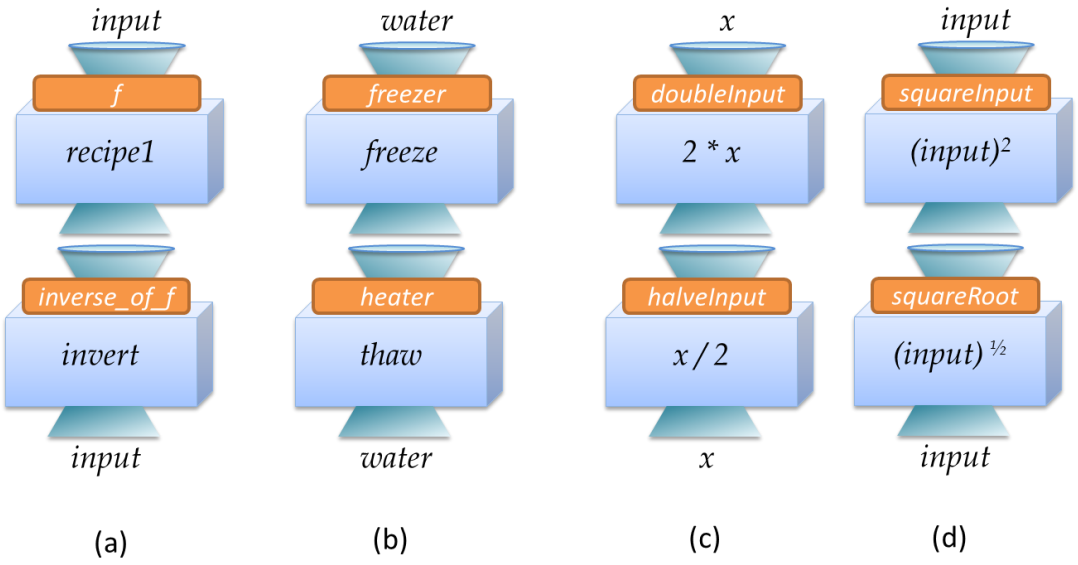
We looked at a function as a model producing an output (say, y) for a given input (say, x). If we start with that output (y), is there another function that would revert it to the input (x) we started with?

Why is this important in the real world? Think about a natural process. Let's take freezing as an example. If we freeze water (input), we get ice as the output. We can reverse this process (function). If we heat up (thaw) ice, it will become water again. Some processes are reversible, and for such processes, it is important to know what the reverse process is. This reverse process is called the **inverse**.

All processes (hence functions) don't have an inverse. Take our very first example. If we give dogfood to a dog, she can produce poop. But there is no process in the world to make dogfood using poop as input!

However, many important functions you find in practice have inverses. For instance, a convex lens produces an upside-down image (when the object is far enough). You can use a second lens to invert that image back to get an upright image.

The figure below visualizes the inverse of a function using our Visual Model. Figure (a) shows the general description of inverse. If you feed in an input to a function and get an output, and then feed that as an input to the inverse of that function, you get your original input back.



Notice that we are doing function composition here (as we learned in Section 1.10). We are feeding in a function into its inverse, to get the original input back.

Figure (b) shows function "freeze" and its inverse "thaw", as we described before. Figure (c) shows an algebraic example. If we send an input x through the doubleInput function, we get the output $2x$. Then, if we send that $2x$ through its inverse, halveInput, we get our original input x back. Therefore, doubleInput and halveInput are inverse of each other.

Notice that I said "inverse of each other". Why? Because you can do this the other way around too. If you send an input x through halveInput first, and feed that output into doubleInput, you get your original x back. This is a special case, because earlier we saw that order matters when it comes to function composition. Inverse is a special case of function composition, where order does not matter. However, as they say in TV commercials, "some restrictions apply", as we will soon find out.

Figure (d) is another algebraic example. If we square an input first and then feed that output into square root function, we get our original input back, as long as we are dealing with non-negative numbers. You can also reverse the order. If you apply squareRoot first and then squareInput, you will get the original number. This will work for all non-negative numbers.

Computer code for `doubleInput` and its inverse `halveInput` is shown below. At the end, we use function composition as we did before by feeding a function into its inverse. For instance, to calculate `out1`, we evaluate `halveInput(10)`, which is 5, and then feed that into `doubleInput`, which evaluates to 10 again.

```
function doubleInput( number1 )
{
      output = 2 * number1
      return output
}
function halveInput( number1 )
{
      output = number1 / 2
      return output
}
```

```
out1 = doubleInput( halveInput( 10 ) )
print( out1 )                    // prints 10

out2 = halveInput( doubleInput( 10 ) )
print( out2 )                    // prints 10
```

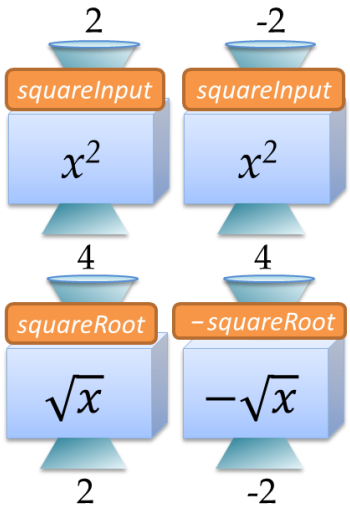
If you have difficulty understanding the function composition in the computer code, I have given the expanded form below, with a temporary variable y to represent the output of halveInput(10).

```
y = halveInput( 10 )      // y = 5
out1 = doubleInput( y )   // out1 = 10
print( out1 )
```

2.2.1 *When Functions Refuse to Invert*

Now we are ready for the fine print. We defined squareInput and squareRoot functions in figure (d) above for only non-negative inputs. This is because, if we use -2 to as the input to

the squareInput function, it will produce 4, which makes the squareRoot function produce 2 as its output. That's not our original input. Why does this happen?



The problem starts with the squareInput function itself. For two separate input values (e.g., 2 and -2), it produces the same output, 4. This is perfectly OK for a function. However, now, its inverse function, squareRoot, faces a major difficulty. If squareRoot function sees 4 as input, should it produce 2 or -2 as output? It cannot produce both. That's not allowed for a function, as we saw in Section 1.2. A function cannot produce two different outputs for the same input. So, we have to restrict the input domain of the squareInput function to either positive or negative values. For positive input values of squareInput, the inverse of the squareInput function is positive squareRoot. For negative input values of squareInput, the inverse of the squareInput function is negative squareRoot (i.e., square root multiplied by -1), as shown in the Visual Model.

If this seems confusing, let's look at a real-world analogy. Let's say that there is one road to your house. So, if someone arrived at your house, you know exactly how he got there. Thus, you can tell him exactly how to go back — just follow the road in the reverse direction. That's the inverse operation. Now, assume that there are two roads to your house. If someone is at your doorstep, you don't know exactly how she got there (she could have taken either road, or even both roads if she were a quantum physicist!). So, you cannot tell her exactly how to get back to where she came from (you cannot find the inverse). Say you are really mad about this situation (who wouldn't be?). What can you do? You go and block one road to your house, so you know exactly how someone got to your house. This is exactly what we do by restricting the input domain. We close up all paths except one.

The bottom line is this: if a function produces the same output for two or more inputs, before finding its inverse, we have to restrict its input domain to some subset of input values. For this subset, the function must not produce the same output value for two different input values.

2.2.2 Finding Inverse Functions

In figure (c) above, we looked at function doubleInput and its inverse. However, how do you find the inverse of a given function algebraically? Let's start with function $f(x) = 2x$. Let's use a variable y to represent the output of f, so we get:

$$y = 2x \qquad \text{[multiply input by 2]} \qquad \textbf{(1)}$$

We know that the above is a relationship between x and y, where x is the input and y is the output. To find the inverse, we want to find the relationship where y is the input and x is the output (i.e., switch input and output). How do you do that? Pretty easy. Use the same relationship and solve for x (because, now x is the output). Then from (1) we get:

$x = y / 2$ [divide input by 2] **(2)**

That's the inverse of (1). Notice that we calculated inverse by solving for the input.

We can calculate inverse by solving for the input (x)

Also notice that (2) still has the same relationship between x and y as in (1) — only the output and input are switched. Thus, in (2), y is the independent variable and x is the dependent variable. Therefore, what we have is a function of y:

$g(y) = y / 2$ [divide input by 2] **(3)**

There are a couple of things to note. First, instead of using a separate function name g, to identify the inverse of f, we usually use the name f^{-1}. Notice that the superscript -1 is just a part of the name, indicating that it's the inverse function of f, the same way a child of a person named John Miller could be called John Miller Jr. If this is confusing to you, always think of f^{-1} as $f^{\text{inverse}}(x)$ or $f\_inverse(x)$, where we use the word "inverse" instead of superscript -1. In fact, I prefer f^{inverse} or $f\_inverse(x)$, because students tend to confuse $f^{-1}(x)$ notation with x^{-1}, which is $1/x$. Therefore, it is good to have the following mental picture:

$$f^{-1}(x) = f^{\text{inverse}}(x) = f\_inverse(x)$$

Second, as always, the input variable name can be anything. Therefore, (3) can be written as $g(u) = u / 2$, $g(x) = x/2$, etc. The name of the input variable does not change the relationship. However, it is customary to use x as the independent variable so you will often see (3) as:

$f^{-1}(x) = x / 2$ [divide input by 2] **(4)**

It is important to note that both (3) and (4) represent the same relationship. We just halve the input to produce the output. The function name and the input variable can be anything.

Let's confirm that the inverse relationship holds between f and g. That is, if we feed the output of f into g, we should get the original input back. That is, we should have:

$$g\,(\,f(x)\,) = x \qquad \text{[function } f \text{ feeding into function } g \text{]}$$

Let's see whether this is the case:

$$\begin{aligned} g(\,2x\,) &= (2x)\,/\,2 \qquad \text{[function } g \text{ divides input by 2]} \\ &= x \end{aligned}$$

Similarly, if we feed the output of g into f, we should get the original input back. That is:

$$\begin{aligned} f(\,g(x)\,) &= 2 * g(x) \qquad \text{[function } f \text{ multiplies input by 2]} \\ &= 2 * (x/2) \qquad \text{[function } g \text{ divides input by 2]} \\ &= x \end{aligned}$$

This confirms the inverse relationship between f and g.

2.2.3 Using Inverse Functions to Solve Equations

In the previous section, we saw that we can calculate the inverse of a function by solving for its input. This is a very powerful observation. We were hinting about such a connection in Section 2.1 while solving equations.

This observation leads to a practical application of inverse functions. The inverse function lets you find the solutions to equations of the form

$$f(x) = b \qquad \text{[} b \text{ is the output for input } x\text{. E.g., } f(x) = 14 \text{]}$$

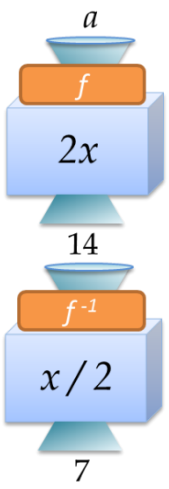
where b is a constant. For instance, take function

$$f(x) = 2x. \qquad \text{[function } f \text{ multiplies input by 2]}$$

Now, say that for some input value, a, we got the output value 14. This means:

$$f(a) = 14 \qquad \text{[input value } a \text{ produces output 14]}$$

Which value of a would give us the output of 14? How do you find that out? Of course, we can use the inverse function of f. We just feed the output of f, 14, as the input to the inverse of f. The inverse of function f is f^{-1}. So, we have to evaluate $f^{-1}(14)$. What is $f^{-1}(14)$, by the way? From (4), we have:



$$f^{-1}(x) = x / 2 \qquad \text{[divide input } x \text{ by 2]}$$

Then, $$f^{-1}(14) = 14 / 2 \qquad \text{[divide input 14 by 2]}$$

$$a = 7$$

This is a very important observation. If you know the inverse function of $f(x)$, you know the solution to the equation of the form $f(x) = b$. Therefore, finding a solution and finding the inverse follow the same steps.

Inverse function, f^{-1}, can be used to solve the equation $f(x) = b$

You can also derive this relationship algebraically as follows. Let's say the output of the function f for input x is b. In other words, we have

$$f(x) = b \qquad \text{[} b \text{ is output of function } f \text{ for input } x \text{]}$$

Taking the inverse of both sides (i.e., doing function composition with f^{-1} on both sides), we get:

$$f^{-1}(f(x)) = f^{-1}(b)$$

Notice that the left side is feeding x into a function f and then feeding its output into its inverse, which gives back x. So, we have:

$$x = f^{-1}(b) \qquad \text{[} x \text{ is the output of function } f^{-1} \text{ for input } b \text{]}$$

Therefore, we can solve $f(x) = b$ by using f^{-1}.

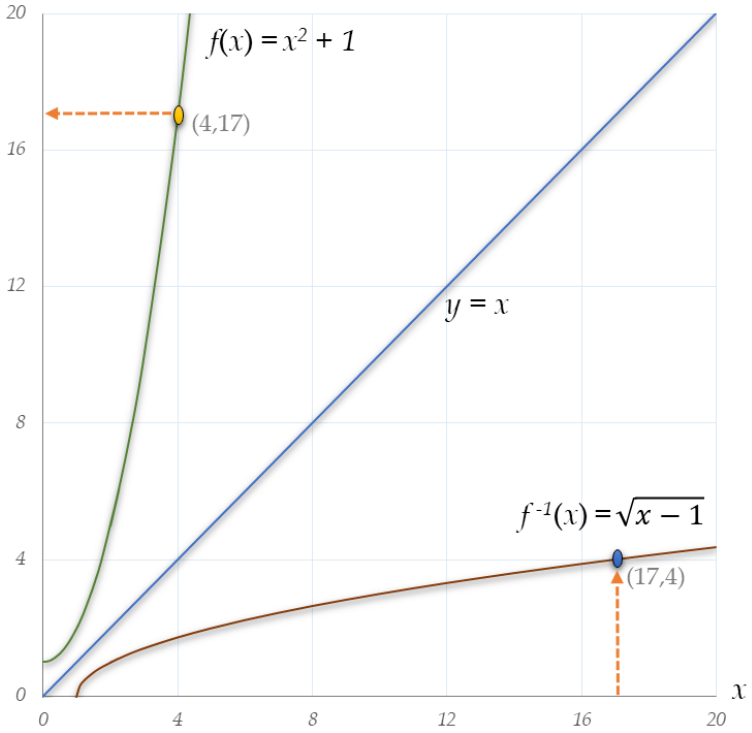
2.2.4 Graph of Inverse Functions

Graphs are useful in visualizing input and output relationships. Since a graph is a complete representation of all input and output values, we can use it to find both:

- the output value for a given input, and
- the input value for a given output.

In other words, a graph is useful for solving equations (finding input for a given output), in addition to finding output for a given input (function evaluation).

For instance, let's take the graph of function $f(x) = x^2 + 1$, for non-negative inputs, as shown by the green curve. If we wanted to find the input value that produces output 17 (the yellow dot), we see that the corresponding input is 4 – i.e., the yellow dot is point (4, 17).



Notice, that we just solved the equation $f(x) = 17$, or $x^2 + 1 = 17$, for non-negative inputs, using the graph of f. In other words, we found that $f(4) = 17$, and according to the definition of inverse function, we see $f^{-1}(17)$ is 4. We can plot this point on our graph too (blue dot). If we do that for all output values of f, we get graph of f^{-1} function, which is shown by the brown curve.

Notice that if point (4, 17) is on the green curve for $f(x)$, the point (17,4) is on $f^{-1}(x)$. Why? We know that if y is the output of f for input x, then point (x, y) is on the curve of f. Now, if we feed y as input to f^{-1}, according to the very definition of an inverse function, it outputs x – i.e., the original input to f. Therefore, point (y, x) is on the curve of f^{-1}.

Graphically, this switching of input and output between f and f^{-1} leads to an interesting result. That is, f^{-1} is a reflection (mirror image) of f, if we keep a mirror at the line $y = x$ (blue diagonal line). Therefore, the blue and yellow dots above are mirror images of each other – i.e., they have the same distance to the blue diagonal line, similar to an object and its reflection in a mirror.

Graph of f^{-1} is a reflection of graph of f (with mirror placed at line $y = x$)

Therefore, given a graph of a function, you can derive the graph of its inverse without additional calculations. You just have to reflect it using a mirror placed on line $y = x$.

2.2.5 *Inverse Operators*

Now that we know how to find the inverse, can you find the inverse functions of common operations? Let's start with addition. If we add 5 to an input x to produce output y, how do we get x back? We subtract 5 from y. Therefore, addition and subtraction are inverse operations. Similarly, what is the inverse of multiplication? If we multiply 5 by an input x to get output y, how do we get x back? We divide y by 5. Therefore, multiplication and division are inverse operations.

Here is a surprise. You may have been applying function inverse even without knowing. When solving an equation, you were taught to do the same operation on both sides of the equation. For instance, if you have to solve the equation $2x = 10$, you would divide both sides by 2. Why were you taught to do that? You may say, we want to isolate x on the left side. Yes, but why would you divide? That's because $2x$ is a result of multiplication, and you need to apply its inverse (i.e., divide) to revert that multiplication. So, here is another way to think about the original equation, assuming we have a function called multiplyByTwo that multiplies its input by 2 and another function divideByTwo that divides its input by 2:

multiplyByTwo(x) = 10 [original equation $2x = 10$]

Now, apply inverse of multiplyByTwo, which is divideByTwo, to both sides:

divideByTwo(multiplyByTwo(x)) = divideByTwo(10) [apply inverse]

x = 5

So, when you were "solving for x" in an equation by doing the same operation on both sides, you were actually applying function inverse every step on the way! Therefore, after you have applied those inverse operations and "solved for x", you are left with the inverse function! So, it is not at all surprising that we can use the inverse function to "solve for x".

Now, here is a puzzler. Are there functions whose inverse is itself? Multiplying by 1 is an example. When you multiply x by 1 you get x itself. So, if you do the same operation again, you get the original number x. That's a trivial case. There is a more interesting case: multiplying by -1. If you multiply input x by -1, you get output $-x$. If you multiply that output $-x$ by -1, you get the original number x.

Why are we so interested in function inverse? It's because, in practice, you often need both a function and its inverse. For instance, if you have a route to get to work or to school, you need a way to get back home too. Algebraically, if a given input produces an output, often we need to find out what input produces that output. That's why we need function inverse.

You will also find that some operations are easier to do after we take the inverse (i.e., in the inverse domain). For instance, when you are reading an article on a computer or a smartphone, often you need to zoom-in. It is easier to see details after zooming in. However, once you are done looking at the details, you need to zoom-out to continue to read normally. Zoom-in and zoom-out are inverse operations of each other. Similarly, if you are standing up and want to put on shoes, it is really awkward to do so standing up. You would first sit down, then put-on your shoes, and then stand up again. Sitting down and standing up are inverse operations. Putting on shoes is easier after sitting down. Therefore, in many fields we often move between a mathematical model and its inverse. Such operations and their inverses are usually called forward "transformations" and reverse/backward/inverse transformations, respectively, and some operations are easier after you do a forward or an inverse transformation. If you hear fancy names like Fast Fourier Transform (FFT), Laplace Transform, Discrete Cosine Transform (DCT), you would be right to guess that they have inverse transformations (operations) as well. For those of you who are familiar with calculus, differential and integral operations exhibit the same inverse relationship.

If you did not pay much respect to function inverse, I hope I changed your mind. In practice, the inverse is as important as the original function itself! They both go hand in hand.

2.3 Putting it All Together: Solving Equations

In this section, we are going to put all the concepts we learned before to solve equations. Remember, solving equations is just another fancy way of saying finding input for a given output. Before we do that, we need to visit just one other concept.

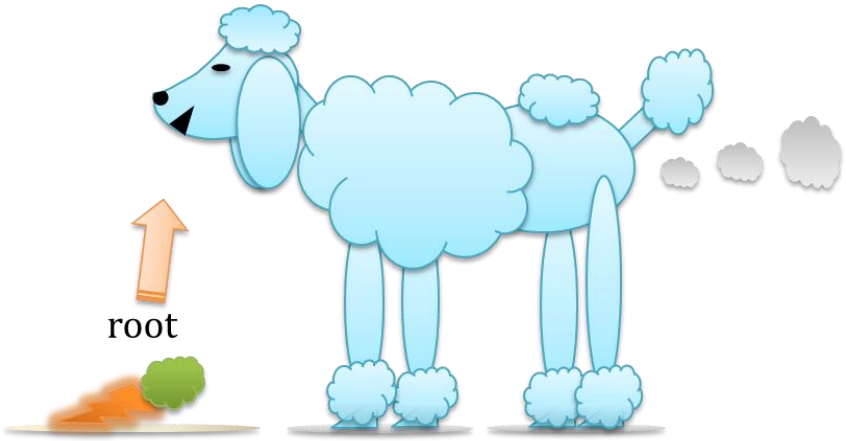
2.3.1 Bad Root! You Produced Nothing!

A function produces output for any input. But all inputs are not created equal. Some inputs lead to special outputs.

Zero is a special output. It means nothing. But what's more important is what input leads to this nothingness. Think about it for a moment. A function's job is to produce an output. The function could do an elaborate manipulation of its input to produce an output. For instance, say you have a recipe to make a cake. It has many ingredients (inputs) and a complicated recipe to produce a cake. Now, if I told you that there is some input for which you will not get any cake, you will be surprised. It must be a really strange input, that may say something special about the function itself.

Let's take an example. The length of your shadow can be modelled as a function. When the sun is out, the length of your shadow due to sun is determined by the angle of the sun. When would the length of your shadow become zero? When the sun is directly above your

head (assuming your body is shaped like a cylinder!). This immediately tells you some important thing about this function. Unless, you can get directly under the sun, you will always cast a shadow. In other words, if you live in a place where the sun does not shine directly above your head, you will always cast a shadow when the sun is out.



Bad Root! You produced nothing!

An input that produces zero as its output is called a **root** (such inputs are also called "**zeros**", not surprisingly). We can always re-arrange an equation to give an output of zero. For instance, take the equation:

$$5x^2 = 2x + 3$$

For instance, the left side ($5x^2$) could represent the path of a comet in a given 2D plane, and the right side ($2x + 3$), could represent the path of a probe we launch in space, in the same plane. We want to find out where (and whether) they would intersect. In other words, if you were to graph this, the left side ($5x^2$) represents a parabola and the right side ($2x + 3$) represents a straight line. The above relationship (equation) is satisfied when both sides produce the same value — that is, when the probe and the comet intersect.

However, rather than finding when both sides would produce the same value, we can re-arrange the above equation so that right side is zero:

$$5x^2 - 2x - 3 = 0$$

This is an equation of the form $f(x) = 0$, where $f(x) = 5x^2 - 2x - 3$. In other words, we transformed the problem to a new form, where we have to find the input values of x, that makes $f(x)$ zero. Simply put, we have to find roots of $f(x)$.

Depending on how many roots $f(x)$ has, we can determine how many times the comet and the probe would intersect. If it has no real roots, there are no points of intersection. If there

is one root, there is only one point of intersection. Similarly, if there are two distinct roots, they intersect at two distinct points.

2.3.2 ROOTING FOR THE INVERSE

The above example shows that, when we have to find input values that make the output of $f(x)$ zero, we just have to find the roots of $f(x)$. These roots are the "**solutions**" to the equation $f(x) = 0$, because the roots satisfy that equation. That is, finding solutions to the equation $f(x) = 0$ and finding roots of $f(x)$ are the same thing.

Solving $f(x) = 0$ is the same as finding the roots of $f(x)$

In general, if we have to solve equation $f(x) = b$, instead of equation $f(x) = 0$, we just have to find the roots of $f(x) - b$. For instance, to solve $5x + 3 = 2$, we have to find roots of $(5x + 3) - 2$, or roots of function $5x - 1$.

How do you find roots of a function $f(x)$? In other words, how do you solve the equation $f(x) = 0$? That's not always trivial. We saw one way to do that in a previous section, with function inverse. If we know the inverse of a function, we can find the solutions to an equation of the form $f(x) = b$. For instance, say if we have, $f(x) = 2x - 4$. Its inverse function, $f^{-1}(x)$ is $(x + 4)/2$.

So, to find the values of x, which satisfies equation

$$f(x) = 0 \qquad \text{[function } f \text{ produces output 0 for input } x \text{]}$$

we can take the inverse of both sides to produce:

$$x = f^{-1}(0) \qquad \text{[function } f^{-1} \text{ produces output } x \text{ for input 0]}$$

Since $f^{-1}(b) = (b + 4)/2$, we get [using the fact $f^{-1}(x) = (x + 4)/2$]

$$f^{-1}(0) = (0 + 4)/2$$
$$= 2 \qquad \text{[function } f^{-1} \text{ produces output 2 for input 0]}$$

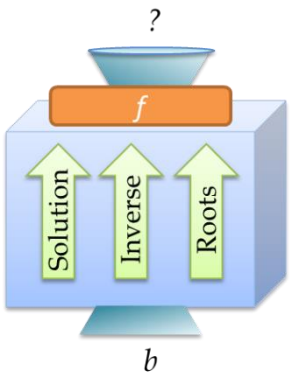
The above means that $f(2)$ is zero. That is, 2 is a root of function $f(x)$, which means

$$f(2) = 0 \qquad \text{[function } f \text{ produces output 0 for input 2]}$$

Therefore, finding the roots of a function f is the same as evaluating f^{-1} at input value 0, if f^{-1} exists.

$f^{-1}(0)$ gives us roots of $f(x)$, or solutions to $f(x) = 0$

However, as we saw previously, f^{-1} does not always exist. In such cases, we need to resort to other methods to find the roots of a function, in order to solve equations of the form $f(x) = 0$. Many polynomial functions in particular, do not have inverse functions and we will see a different method to find their roots in Section 4.2.1.2.



To summarize, when we know the output b of function $f(x)$ and want to find the input value that produces b, we have several options, as shown in the Visual Model. We can

- directly solve $f(x) = b$, [e.g., solve $f(x) = 2$, where b is 2]
- find the inverse of f and then find $f^{-1}(b)$, or [e.g., find $f^{-1}(2)$]
- find the roots of $f(x) - b$. [e.g., find roots of $f(x) - 2$]

All of these achieve the same objective of solving an equation.

Recall from Section 2.2 that directly solving an equation is nothing more than repeatedly applying inverse operators to both sides of an equation.

Now you should be able to clearly relate several seemingly unrelated concepts:

- Finding input for a given output entails solving an equation
- Solving equation $f(x) = b$ is same as finding roots of $f(x) - b$
    - When the inverse function exists, we can use that to directly find roots (or we can apply inverse operators repeatedly to solve for input)
    - When the inverse function does not exist or difficult to calculate, we need other methods to find roots

Now you know the value of roots and inverse functions. They are essential when we need to find which input produced a given output in mathematical models we build.

Finding solutions for mathematical models is a vast and complicated subject area. In the coming chapters, when we explore a new model, we will take effort to look at its inverse model (if it exists) or other root finding techniques that allow us to solve for input.

2.5 The Story So Far

As we mentioned before, this is a story about the function dynasty. In the first chapter, we looked at how functions came into existence, why they are such experts in building relationships, and what properties they have in common. This chapter was dedicated to a common problem they face and some common tools they can use to solve that relationship issue. In the process, we explored several key concepts.

First, we looked at the problem the functions faced every day. Being members of a dynasty, they don't have usual money and employment problems that ordinary folks have. Their problem is rather unique. If they are confronted with an output they have produced, it is not always straightforward to figure out which input they consumed led to that output.

Second, we discovered how this problem requires solving equations. When we know a function and its output but do not know which input produced that output, that leads to equations of the form $f(x) = b$, where b is the output and x is the input. We need to solve such equations to find out which input produces the given output.

Third, we looked at inverse functions. As a particular example of composition, we saw that feeding the output of a function to its inverse function allows us to get the original input back. This is invaluable in the real-world. In a big family, if there is someone who does something, there is another who undoes it. If there is a guy who makes jewelry from gold, there is another guy who melts jewelry to get back gold. The function and its inverse go hand in hand, and inverse functions are quite useful in solving equations of the form $f(x) = b$.

Fourth, we discovered that some functions produce the special output value of zero for certain inputs. These inputs are called roots, or zeros, of the function and are solutions to equations of the form $f(x) = 0$. We can use inverse functions, when they exist, to find roots, and hence solve equations. That enables us to find the input for a given output, for the mathematical models we build.

Having studied what traits and problems functions have in common, in the next chapter we are going to look at another common aspect that is really important to them: *what* they consume and produce.

3 Numbers: What Functions Consume and Produce

Chapter Overview: In this chapter, we will gradually build up the concept of numbers, starting from whole numbers. We will see how using basic arithmetic operators on whole numbers leads to new numbers. Finally, we will explore a new operator, exponentiation.

If functions are recipes, numbers represent the ingredients that go into and the final result of the recipe. They are the mathematical representations of objects in the real-world.

Numbers were invented out of sheer necessity. Can you imagine how people communicated before numbers were invented?

Why cavemen did not have supermarkets

Numbers **represent** how many of a kind there is. They are a very powerful abstraction. The same number 5 can represent 5 apples, 5 oranges, or 5 of any kind. It took a while for humans to grasp that a number (a symbol), say 5, can represent some property about objects as diverse as apples, cows, trees, monkeys, etc.

3.1 Invasive Numbers: How Numbers Breed More Numbers

People first used numbers like 1, 2, 3, 4, … for counting (e.g., for counting chickens). They are called counting numbers, or **whole numbers** (We also call them positive numbers or natural numbers). They are pretty useful for our everyday lives.

Yes, whole numbers are useful, but are they really sufficient for all our purposes? For instance, to model real-world situations, we want to add, subtract, multiply, and divide numbers. So, let's try those four basic arithmetic operations using these whole numbers and see how they stack up to our requirements. Let's not forget that arithmetic operations are functions, as we saw in Section 1.9. So, basically, we are going to see what kind of outputs we get with our four basic functions, if we use *whole numbers* as inputs.

3.1.1 Addition and Multiplication

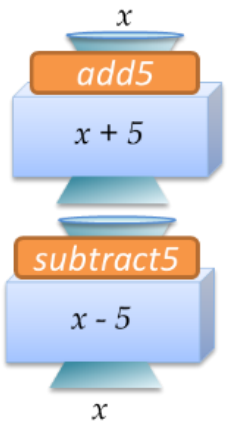
Adding 5 apples to a pile of 10 apples will make the pile 15 apples. Pretty easy, eh? Even a caveman can grasp the concept.

What about multiplication. You may have been taught multiplication as repeated addition. Later, we will discover why this is not exactly true. But for the moment, if you have to find the total number of apples in 3 bags, where each bag contains 5 apples, 5 + 5 + 5 works fine.

3.1.2 First Signs of Trouble: Subtraction

Subtraction can be thought of as finding the difference between two numbers. A nifty way to think about subtraction is as follows. If we want to evaluate 5 – 3, we can think of this operation as finding what needs to be added to 3 to get to 5 (i.e., 2 needs to be added to 3 to get to 5).

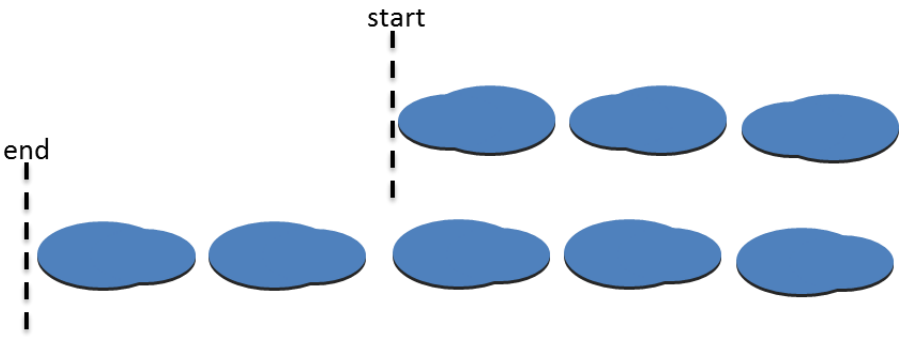
As we saw in Section 2.2.5, subtraction is the inverse function of addition. As an example, consider the function for adding 5 to the input and its inverse function, which is subtracting 5 from the input, as shown in the Visual Model.



Subtracting 3 from 5 is dandy, but what about 3 – 5 (subtracting 5 from 3)? What does this really represent? If we had 3 cows and 5 cows died due to an illness … wait a second. How can 5 cows die if we had only 3 to begin with?

Is 3 – 5 really nonsensical? The answer is both yes and no. The answer depends on what each number represents.

If the numbers represented cows, of course you cannot subtract 5 cows from 3. However, let's say we represent a forward step with the number 1. Then, if we took 2 steps forward and took 1 step back, we can represent that with 2 – 1. Accordingly, 3 – 5 represents "3 steps forward and 5 steps back", as shown below. This results in "2 steps back" from the position we started.



If numbers represented objects that have a "**sense**", like forward and backward, where one "sense" is opposite of the other, the expression 3 – 5 is a perfectly valid expression. However, if the numbers represented objects without any such "sense", then the expression 3 – 5 has no meaning. This is a subtle but a very important point. Rather than blindly applying algebra, we need to first think about what each number represents. Then and only then, can you really understand a mathematical model.

The above example shows that whole numbers are not enough to represent real world objects. We need to expand the definition of numbers to include a "sense" (Note that I did not use the word "direction" instead of "sense" because "direction" has a more general meaning like east, west, north, northwest, etc.)

Whole numbers are not enough to represent objects in the real world

With this new expanded definition, numbers can represent 3 steps forward or 3 steps backwards; 3 steps up or 3 steps down; 3 steps above or 3 steps below. To distinguish one sense from its opposite sense, we need another symbol. For that purpose, we use a **sign** (either + or -). With this, we can represent three steps forward by +3 and 3 steps backwards by -3. The positive sign is usually omitted for brevity.

It is unfortunate that we use the same symbol for both addition and positive sense (and subtraction and negative sense). It is important to note that 5 + (-3) should be read as "five plus negative three".

Note that which sense is positive is quite arbitrary. We could represent a "forward step" using -1 and a "backward step" using +1. In that case, 3 steps forward and 5 steps backward (as in the example above) has to be represented by (-3) + 5. This results in +2, which means "2 *backward* steps" in this case. In either case, you end up "2 steps backwards" from where you started. However, if you change the sense, to represent the

same real-world situation, you have to modify expressions. The final result in the real-world is unchanged.

The same way subtraction led to negative numbers, subtraction leads to another number we have not seen in this section so far: zero. For instance, 5 – 5 leads to zero.

Subtraction of whole numbers leads to negative numbers and zero

3.1.3 More Trouble: Division

You may have been taught to treat division as dividing a given number of objects into a given number of groups. That is, dividing 12 apples among 3 groups results in 4 apples per group.

As we saw in Section 2.2.5, division is the inverse function of multiplication. Consider function f that multiplies its input by 3 and function g that divides its input by 3. Function f and g are inverses of each other.

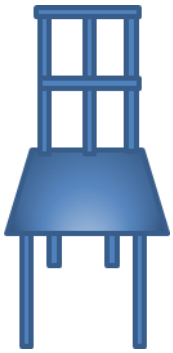
This is all dandy, but trouble starts when we divide a whole number by another whole number that leads to a non-whole number: a number we have not seen up to now. If we divide 5 apples into two groups, each group ends up with two whole apples and half an apple. That is, five divided by two gives us two and a half, which cannot be represented by any whole number we have seen so far. So, we need a new number — i.e., a fraction (5/2) or a decimal fraction (2.5).

Division of whole numbers leads to fractions

3.2 Unwholesome Whole Numbers

What's really wrong with whole numbers? As we saw above, both subtraction and division of whole numbers can produce new numbers that are not whole. It seems like the concept of whole numbers is really inadequate (or broken, if you may). But why? The answer, as we will soon find out, is quite enlightening.

3.2.1 Poor Abstraction of Reality

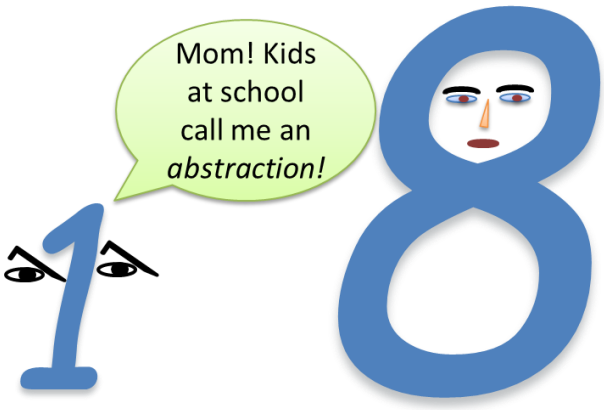


As we discussed before, numbers are an abstraction of real-world objects. But before we go any further, we need to understand what an **abstraction** is. The figure here shows an abstraction of a chair. We model it with 3 features. It has four legs to stand on, a flat surface to sit on, and a back-support to lean on. That's it. It can represent any chair with these 3 features — e.g., a wooden chair, a sofa, a desk chair, an office chair, or a chair in a diner. But what about a "swiveling chair"? The abstracted chair cannot represent "swiveling". Similarly, it cannot represent the "foldable" property of a folding chair nor can it represent the "reclining" property of a recliner. The bottom line is this: an abstraction represents some properties of real objects, but does not represent some other properties of real objects.

As we discussed before, numbers are an abstraction of real-world objects. So are operators, which are functions used to model real-world relationships. If that is so, how come applying operators to whole numbers as inputs produce "un-whole" numbers?

The answer lies in the quality of abstraction (i.e., how many features are abstracted, as we saw with our abstraction of the chair). Every abstraction is not created equal. As we will see soon, operators are a better (or richer) abstraction of reality compared to whole numbers.

We know that real world objects do not come as just whole numbers. There can be fractions (e.g., half of an apple), and negative numbers (5 steps forward and 8 steps backwards). Whole numbers cannot represent these real objects. Thus, whole numbers are an inadequate abstraction of real objects. However, that's not a real surprise. How did operators (functions) reveal that whole numbers are not the real deal?



Operators (functions) reveal that whole numbers are not the complete picture because operators are better abstractions of real-world relationships. Take division as an example and imagine the real-world scenario of dividing a loaf of bread among 3 people (i.e., 1 divided by 3). That would force us to ask why a whole number cannot represent the output of our divvying up of bread. It also forces us to think about a new number that *could* represent the output of this division. Similarly, if we define subtraction as "finding difference", in the real world we can always look at the difference between two positions — for instance, where would we end up if we take 3 steps forward and 5 steps backwards? This is the difference between the end position and the starting position. This forces us to come up with a new number that can represent this output.

It's fascinating how functions exposed whole numbers as a poor abstraction of the real world and pointed us in the right direction. In Chapter 6, we will see functions doing the same thing again. All in all, now we have whole numbers, negative numbers, fractions, and zero. All of these numbers are collectively known as **rational** numbers.

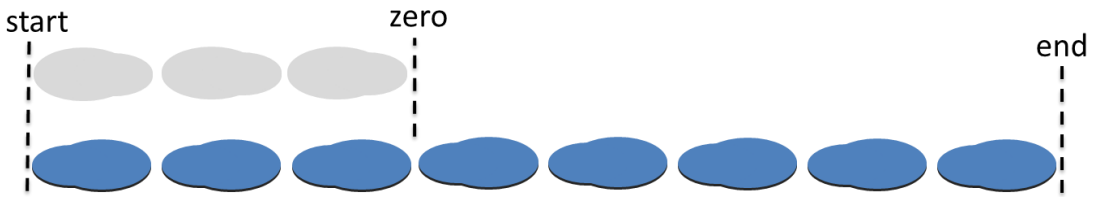
3.3 Operations Revisited (with Negative Numbers)

Let's look at the same four basic arithmetic operations, but this time with rational numbers (especially with negative numbers).

3.3.1 Addition and Subtraction

Adding a positive number to a negative number can be easily understood using "sense". For instance, 5 + (-3) can be interpreted as "go five steps forward", and then "go three steps backwards".

Subtraction can be thought of as finding difference. For instance, 5 – 3 can be thought of as: how many apples are needed to make it to 5, if we start with 3 apples? Therefore, 5 – (-3) can be thought of as follows: if we start at a position that is 3 steps behind zero, how many steps are needed to get to a place 5 steps ahead of the starting position? As shown below, we need to go 8 steps forward, which means the answer is +8.



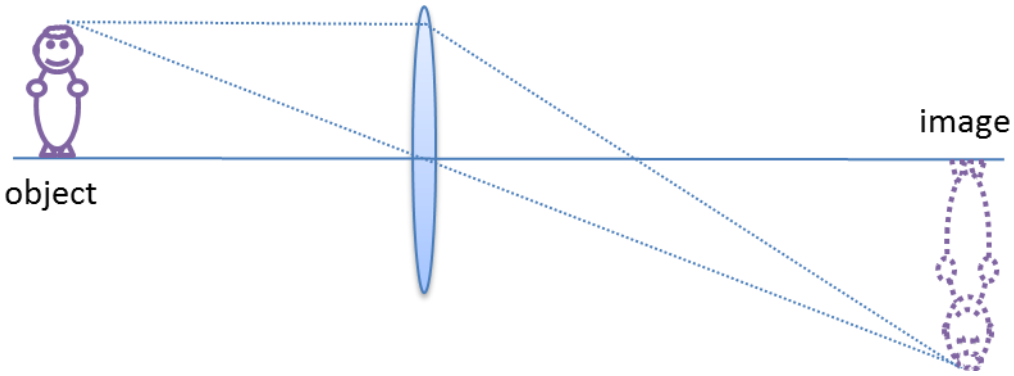
3.3.2 *Multiplication as Two Operations*

If we multiply 3 by -2, we are essentially creating three groups of -2 objects (e.g., groups of 2 backward steps). That would give us -6 (6 backward steps). Repeated addition works fine in this case (i.e., -2 + -2 + -2).

However, what about -3 ∗ -2? You cannot create -3 groups of -2 (or -2 groups of -3). There is no such a thing as -3 groups. Similarly, you cannot repeatedly add -3 negative two (-2) times.

You may have been taught that -3 ∗ -2 = 6 (a Negative times a Negative is a Positive). That's how it works. Period. But that's a very unsatisfactory answer.

Let's start over. The problem appears when multiplying by a negative number. What really happens when we multiply by a negative number? What's a good analogy?

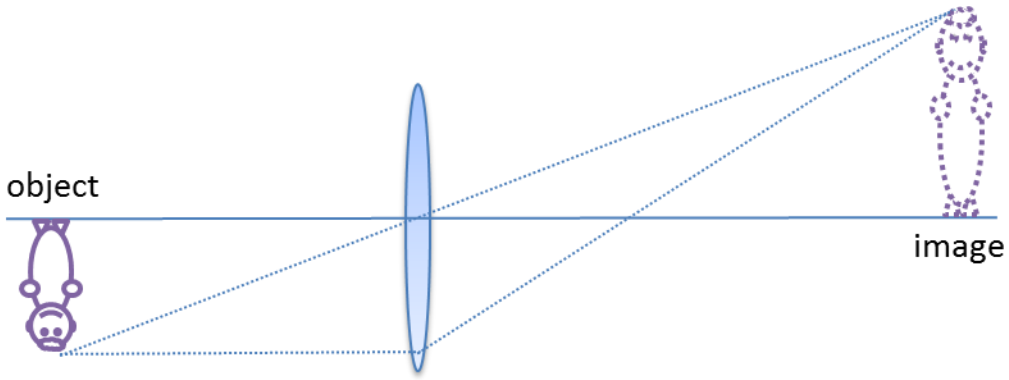


In Section 1.4, we looked at a magnifier as a function. Magnification is similar to multiplying by a positive number. A magnifier is a convex lens. A convex lens can do more than just positive multiplication. It can do negative multiplication as well. That is, when the object is sufficiently far away from the lens, it flips the object, in addition to magnifying it, as shown in the figure above. There, we consider the upward direction to be positive and the downward direction to be negative. Therefore, if the original object is upright (positive), the resulting image is upside down (negative), as shown.

A convex lens can do both positive and negative multiplication

This is an excellent example of multiplying by a negative number because "negative multiplication" does both of these things — it both magnifies a number and flips its "sense". Ok, so what about multiplying a negative number by another negative number? How do we use a negative object in our analogy in the above figure? Well, what's a negative object, according to our definition of positive and negative? An object that is upside down. So, if we

were to start with an upside-down (negative) object, the resulting image would be upright (positive)! We multiplied a negative number by another and got a positive number. The following figure illustrates this clearly.



Computer code corresponding to negative multiplication is given below. As an example, it multiplies by -10. It is quite simple.

```
function multiplyByNegative10( input )
{
        result = -10 * input
        return result
}
```

Even if you don't have a convex lens, you can observe another type of negation (flipping) with an ordinary mirror. In this case, the image does not flip upside down. If it did, when you went into your bathroom, you would see a person brushing teeth standing on his or her head! Instead, what an ordinary mirror does is it flips left and right. If you hold a toothbrush in your right hand, the person in your mirror is holding it in his or her left hand. If you have letter "b" on the front of your shirt, it would appear as letter "d" in the mirror. This is negative multiplication. If we consider the ordinary sense as positive (e.g., sense of "b" is positive), the image ("d") is negative. So, a mirror is essentially a function that multiplies its input by -1. Note that the magnitude is 1 because ordinary mirrors don't make the image larger or smaller, as convex lenses do.

Now, how do you find out what happens with negative multiplication? As with the convex lens, you start with a "negative object" — e.g., a letter that was flipped to begin with. So, the negative input supplied to a negative multiplication operation yields a positive result (i.e., "d" would become "b"). This is exactly why ambulances have the word AMBULANCE written in flipped letters on the front (see figure below). A driver looking through his or her

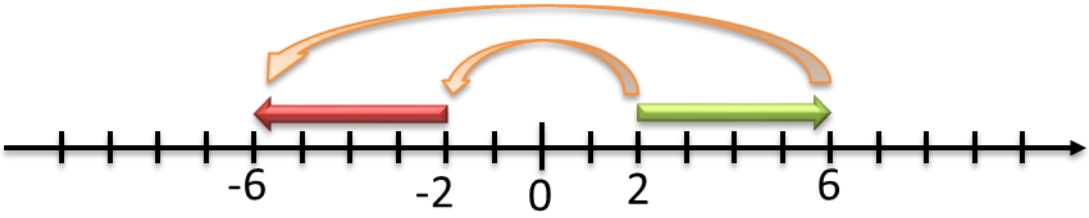
rear-view mirror at an ambulance that is coming from behind, would see the word AMBULANCE correctly. This is an everyday application of negative multiplication.

The above analogies show that multiplication is not just one simple operation. It's actually, two operations in one. We can think of these two operations as:

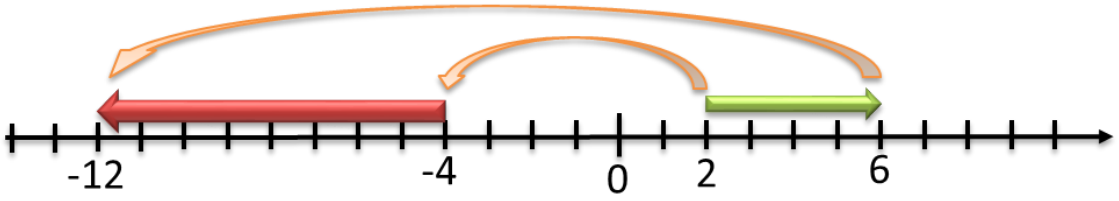
- Magnification (also known as scaling, or amplification)
- Reflection (Also known as flipping, inverting, negation, sense reversal, or rotation by 180^0). This happens when one of the inputs is negative. When one of the inputs is negative, it reverses the "sense" of the other input (i.e., flips the other input).

Multiplication by a positive number can be always thought of as plain magnification or scaling. However, multiplication by a negative number results in reversal of "sense" or reflection, in addition to magnification. A convex lens can perform both operations, by varying the distance to the object.

Now, let's see how this reflection happens with numbers. In the following figure, we look at multiplication by -1. Let's start with the green arrow going from 2 to 6. If we multiply the starting and ending positions of the green arrow by -1, we get the red arrow, going from -2 to -6. As you can see, the red arrow is a reflection around 0. That is, if we were to place a vertical mirror at zero, the red arrow will be the reflection of the green arrow. Similarly, the green arrow is a reflection of the red arrow. The red arrow is from -2 to -6, and if we multiplied these coordinates by -1, we get the green arrow from 2 to 6. This clearly shows how multiplying by -1 acts as reflection (or rotation by 180^0 around zero).



If we consider forward direction of the x-axis as positive, a positive object (green arrow) multiplied by -1 gives us a negative object (red arrow). Similarly, a negative object (red arrow) multiplied by -1 leads to a positive object (green arrow). If we used -2 instead of -1 as the multiplier, it would do magnification, in addition to reflection, as shown below.



Quick quiz: Based on the above discussion, if we wanted a function to model a mirror, what function would you use?

Multiplying by a negative number is both magnification and reflection

The same intuition applies to division. Why? Because, division is the inverse of multiplication. What does that mean? If we send an input through multiplication (say multiply by -5) and if we send the result through division (divide by -5), we should get the original input. Therefore, if multiplication by -5 reflected the original input, division by -5 should reflect it back to produce the original input. As an example, in the above figure, we multiplied the green arrow by -2 to get the red arrow. How would we get the green arrow back from the red arrow? We have to divide the red arrow by -2.

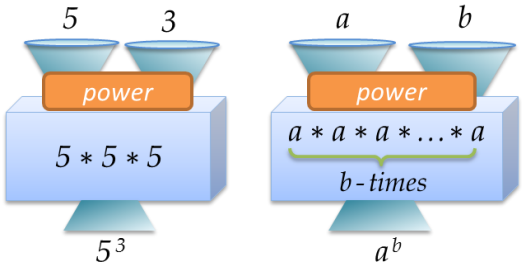
This discussion should help you get a clear intuitive understanding of multiplying by negative numbers (and multiplication in general). Rather than remembering that a "negative times negative is positive", now you have clear real-world analogies to understand multiplication. Further, now you know that multiplication is in fact two operations built into one — magnification and reflection (sense reversal).

In mathematical modeling of the real world, it should be noted that negative multiplication (or division) is meaningful only when the operation we represent can do magnification and reflection (sense reversal). For instance, multiplying 5 cows by -1 would be nonsensical, unless you represent a cow looking forward by +1 and you use multiplying by -1 to model a device that can rotate the cow by 180^0 (similar to a mirror). As with negative numbers, the meaning of the result we get with negative multiplication (or with any other operator or function) depends on the meaning we ascribe to that operation. Algebra could care less about that meaning – it could be used to happily multiply cows, chickens, or apples or anything we ask it. Finding the meaning of that abstract operation is entirely up to us and should be done according to the models we build.

3.4 Go Forth and Multiply, err…, Exponentiate

If we can multiply once, we can do it any number of times, repeatedly! That is called **exponentiation** (or raising a number to a given power). As an example, $5 * 5 * 5$ can be written as 5^3, with 3 as the exponent (power).

Notice that exponentiation is an operator (a function) with two inputs. If we model this *exponentiation operator* as a two-input function named "power", 5^3 could be also written as power(5, 3), as shown in the Visual Model below. Similarly, $x * x * x * x$ is x^4, with an exponent (power) of 4, and can be written as power(x, 4). Instead of using an explicit operator symbol like + or $*$, exponentiation is expressed using superscript to indicate the exponent. There is nothing magical about the superscript; it is just a convention to omit an explicit operator, the same way we omit + sign before a positive number. If it is confusing, you can use the "^" operator explicitly to represent exponentiation, as some programming languages do — e.g., 5^3 can be written as 5^3.



Exponentiation is a function (an operator) — i.e., 5^3 = 5^3 = power(5, 3)

The same way repeated addition leads to multiplication, repeated multiplication leads to exponentiation. As an example, the computer code for producing the cube of an input (i.e., an input raised to the 3rd power) is given below:

```
function cubeInput( number )
{
      result = number * number * number
      return result
}
```

One place you can commonly find exponentiation is in **scientific notation**. For instance, 221 can be expressed as 2.21×10^2 in scientific notation, because 10^2 is 100 and $221 = 2.21 * 100$. The scientific notation uses powers of 10.

There are some consequences of our definition of exponentiation. If we multiply 5^3 and 5^4 together, we get 7 factors of 5:

$$(5 * 5 * 5) * (5 * 5 * 5 * 5) = 5^{(3+4)} = 5^7$$

In other words, the exponents just get added together. Similarly, if you divide 5^4 by 5^3, you get just 5, meaning that exponents get subtracted as shown below:

$$5^4 / 5^3 = 5^{(4-3)} = 5^1 = 5$$

According to the above rule, what is $5^3/5^3$? It is $5^{(3-3)} = 5^0$. However, $5^3/5^3$ is just 1, because we are dividing a number (5^3) by itself. Therefore, any number (say x) raised to power zero is just 1. We can write this as $x^0 = 1$. This is just a consequence of our definition of power function and there is nothing magical about it.

$$x^0 = 1$$

Subtraction of exponents can lead to **negative exponents**: if we divide 5^3 by 5^4, we get $5^{(3-4)}$ or 5^{-1}. Notice that 5^{-1} is equal to fraction 1/5 because 1/5 can be written as $5^0/5^1$, or $5^{(0-1)}$. Consequently, to represent fractions in scientific notation, we use negative exponents: for instance, 0.12 is written as 1.2×10^{-1}.

$$x^{-1} = 1 / x$$

What about **fractional exponents**? If we multiply $3^{½}$ by $3^{½}$, we get 3^1 because exponents add up when we multiply numbers. If we multiply the same number ($3^{½}$) by itself, we square it. That means we get number 3 when we square the number $3^{½}$. This means that $3^{½}$ is the **square root** of 3. Therefore, fractional exponents can be used to express the square root, the cube root, and, in general terms, the n^{th} root of a number.

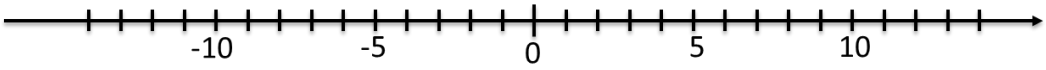
Square root:	$x^{½} = \sqrt{x}$	(e.g., $3^{½} = \sqrt{3}$)
n^{th} root:	$x^{1/n} = \sqrt[n]{x}$	(e.g., $2^{1/3} = \sqrt[3]{2}$)

$x^{1/n}$ is the n^{th} root of x, or $\sqrt[n]{x}$ (e.g., $x^{½} = \sqrt{x}$)

Square roots (and n^{th} roots) of some numbers produce a new set of numbers we have not seen so far. For instance, if we take the square root of 2 (i.e., $\sqrt{2}$), or the square root of 3 (i.e., $\sqrt{3}$), the resulting number cannot be expressed as a fraction. In other words, they are not rational numbers. If we represent $\sqrt{3}$ in decimal notation, there will be an infinite number of digits. We call such numbers **irrational** numbers. Fractional exponents are one way we get these irrational numbers. We could get them by other means too. For instance, π, the ratio between the circumference of a circle and its diameter, is also an irrational number.

Fractional exponents can produce irrational numbers

Rational numbers and irrational numbers taken together form **real numbers**. A real number can represent any number on the number line, which is shown below:



Remember how we got here? We started with whole numbers, that were invented for counting. Then, we saw that, whole numbers could not represent all objects, especially the objects that result from arithmetic operations when we use whole numbers as input. Therefore, we had to come up with fractions, negative numbers, and zero to represent real-world objects. They were called rational numbers. Finally, exponentiation showed us that there are some other objects we cannot represent with just rational numbers, giving us irrational numbers. All of these numbers taken together form real numbers we use every day to represent the real-world objects.

Phew! Finally, we have all real numbers on the number line!

I want you to take a moment to reflect on the approach we took. Rather than drawing a number line and defining all numbers on it as real numbers, we started small and expanded the definition of "numbers" step by step. We started with an abstraction (whole numbers) that was woefully inadequate in representing the real world. We gradually expanded that abstraction until our abstraction became richer, enabling us to represent a whole lot of real-world objects. This process illuminated the intuition behind numbers, and what each type of number can actually represent as well as what it cannot. This process, also forced us to look at our everyday arithmetic operations in a new light. Especially, we got to see the true nature of multiplication.

Ok, are we done then? Not quite. Exponentiation introduced us to fractional exponents — for instance, square roots. However, what if we have to find the square root of a negative number, like $\sqrt{-1}$? We have not yet seen a number w that gives a negative number, when squared. Any number, whether positive or negative, when multiplied by itself produces a positive number. Therefore, there is no real number that can represent the square root of a negative number! This is another prime example of functions (operators) exposing the inadequacy of real numbers (i.e., real numbers are not the "real deal" either). We will see the true nature of numbers in Chapter 6, when we meet complex numbers.

3.5 The Story So Far

As we mentioned before, this is a story about the function dynasty. In the first two chapters, we looked at how functions came into existence, and the properties and problems they have in common. This chapter was dedicated to looking at what they consume and produce. Functions are so successful because of the things they can produce (output) from what they consume (input).

Functions are really savvy consumers. They can easily detect when something they are about to consume is not up to snuff. Operators, which are functions, quickly showed that whole numbers are not the real deal, and showed us how whole numbers are inadequate at representing real-world objects. When we used whole numbers as inputs, functions gave us objects that were not whole numbers, forcing us to expand our horizons, and showed us what numbers should look like.

One particular function, multiplication, turned out to be quite intriguing and versatile. In addition to scaling the input, we saw that it can reflect the input (rotate by 180^0) or reverse its sense. We are still far from discovering all of the tricks multiplication has to offer. In the following chapters, multiplication will show us more of its amazing capabilities.

Just as we discovered all of our real numbers and breathed a sigh of relief, the square root function threw us another curve ball: $\sqrt{-1}$. As we will see in a following chapter, finding a way to solve this problem will again expand our horizons.

Some functions are more useful to us than other functions when it comes to modeling the real world. If you look at your tool box, you are more likely to find screw drivers, pliers, and spanners than, say, a jack hammer, because the former tools are more commonly used for everyday projects. Similarly, if you look at the function dynasty, some members are much more useful than the others. The next chapter is dedicated to those common folks.

4 COMMON RELATIONSHIPS (FUNCTION TOOLBOX)

Chapter Overview: This chapter methodically examines four function families that we meet frequently in our modeling activities in Engineering and Physics. For each family, we will examine its inverse and reciprocal families and finally look at how to combine the members of these families to create more intricate models.

If we look at the real-world, relationships among people are evident everywhere: relationships between parents and children, teachers and students, businesses and clients, people and their pets, and so on. Some of these relationships are more important to us than others. For example, the relationship you have with your parents is more important than the one you have with your hair dresser (we hope!).

Similarly, in math, we can come up with various sorts of relationships. However, some relationships are more fundamental than others. They are more useful to us, and more importantly, they are quite versatile in building real-world models. We are going to study some of these fundamental relationships (functions) in this chapter.

We are going to explore functions in a systematic way. Altogether, we will look at four function families:

1) Power Functions
2) Polynomial Functions
3) Exponential Functions
4) Trigonometric (Periodic) Functions

First, we will look at the first three families, followed by functions that are related to them in two special ways:

a) Inverse of functions in each family: who can undo the effects of a function
b) Reciprocal of functions (i.e., 1 / function) in each family: who produces less output (proportionately) when a function produces more output (and vice versa)

Then, we will look at the 4th family, Trigonometric Functions, along with their inverse and reciprocal functions, separately, in Section 4.6 due to the different treatment they need.

Finally, once we have all our basic tools, we are going to look at how to create more powerful tools by combining them. That is, we will look at how to compose more elaborate functions (models) using basic functions of the above 4 families.

In this book, unlike in many other math texts, functions are also referred to as **models**. This is done intentionally to emphasize why we use functions. We look at functions as tools to build mathematical models of the real-world relationships.

4.1 Power Function Family: How to Grow Your Power

If an input wants to have more power, what can a poor input do? Well, it has one solution. Even ordinary people can wield great power, if they have powerful friends. In algebra, that powerful friend is known by the name of multiplication, who can increase the power of an input twofold, tenfold, or even hundredfold.

If we have an input named x, multiplication can be used to multiply x by itself, any number of times, to create powers of x. For instance, input x multiplied by itself is x^2, and x^2 multiplied by x again is x^3, and that multiplied by x again is x^4. This exponentiation operation, as we saw in Section 3.4, creates the **power function** family. Each member of the family has the ability to multiply its input by itself for a given number of times:

$f(x) = x^1$ [x]
$g(x) = x^2$ [$x * x$]
$h(x) = x^3$ [$x * x * x$]

In general, for a given positive integer *constant* n, we can describe this family as:

$$f(x) = x^n \qquad \textbf{(1)}$$

Again, note that n is a positive integer constant. Let's look at an example: when n=3, we get the power function $h(x) = x^3$. This is one member of the family. We can also express x^3 as power(x, 3), using the more general two-input power function (exponentiation operator) we met before in Section 3.4.

To make our power function a little bit more general, we can multiply the output of a power function by a scalar coefficient, a, to obtain:

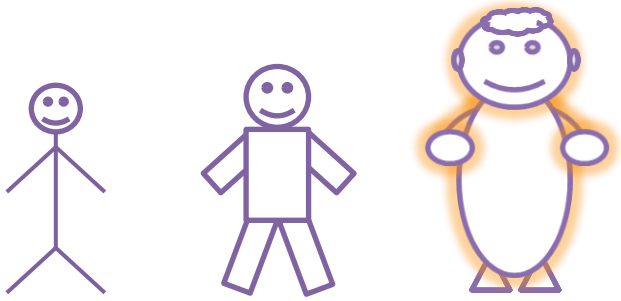
$f(x) = ax^1$
$g(x) = ax^2$
$h(x) = ax^3$

Or, in more general, we can write this as:

$$f(x) = a x^n \qquad (2)$$

When a=1, (2) becomes (1). Notice that the only operator (function) involved in creating a power function is multiplication. For instance, there is no addition operator involved. Additionally, notice that, an expression of the form ax^2 is called a **term,** as we learned in Section 1.5. Therefore, each power function is a term with a different power.

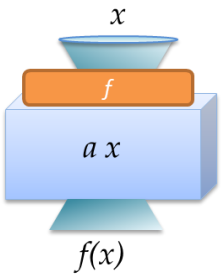
Let's take a closer look at a few important members of this power function family. Again, remember, they are important because they are useful to us and we would need them for creating models of the real world.



Power Function family members with increasing powers

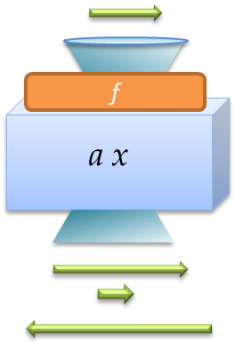
4.1.1 Linear Term

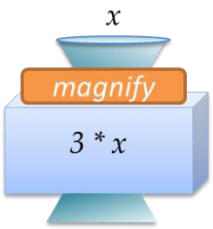
When n=1 in equation (2) above, we get the linear term ax, where a is a constant and x is the input. We use models with linear terms constantly, without even thinking about it. If one apple costs $1, five apples cost $5. That's a linear relationship. In a linear relationship, the output changes proportionately to the input. If we change the input by some amount, the output changes by the same amount multiplied by a constant. It is as simple as that. If the constant multiplier is negative, the output changes in the opposite direction — i.e., when we increase input, output increases in the opposite "sense".



In a linear model, $f(x)$ has a root at zero, because when we use x=0 as our input, $f(x)$ outputs 0. That is, $f(0) = 0$. In fact, zero is the only root. This root leads to some interesting properties, as we will see shortly.

A geometric representation of the function $f(x) = ax$ is given in the Visual Model to the right. If we insert an arrow with some magnitude and "sense" as our input, the possible outputs are given at the output. The output can be scaled-up or scaled-down arrows with the same sense (when a is positive), or the opposite sense (when a is negative).





Let's take a magnifier as an example. Say a magnifier multiplies the height of an object by 3. If an object is 1 inch tall, the image is 3 inches tall. If we increase the height of the object from 1 inch to 2 inches, the image becomes 6 inches (from 3 inches). So, for every inch we increase the height of the object, the height of the image grows by the same number of inches (3 inches). If we double the height of the object, the height of the image doubles too. This is a relationship with a **linear term**.

As a quick quiz, can you explain what the function $f(x) = x$ models, when $a = 1$? Do you need a hint? It's called the "**identity function**". Why? Because its output is the same as (identical to) the input. It doesn't do anything quite useful. It's a trivial relationship.

OK, let's look at a real-world relationship: the flashlight. If you look at a traditional flashlight, it has batteries, an incandescent lightbulb (not LEDs), and a switch. When you turn on the switch, current flows through the lightbulb, making it light up. The thin wire (filament) in the lightbulb offers resistance to the flow of current. Due to this resistance, the filament heats up and glows, producing light. You may have noticed that as batteries become weaker (older), the lightbulb gets dimmer, because the batteries cannot produce enough "pressure" to "pump" current through the thin wire. This relationship is represented by a fundamental law in electrical engineering: **Ohm's Law**.

If you are not familiar with electrical terms, a good analogy is water flowing through a pipe. If the diameter of the pipe is small, it is harder to push a lot of water through it. Thus, the pipe offers "resistance" to the flow of water. Say we want to increase the amount of water (current) that flows through this pipe. Then, we have to apply more and more pressure (voltage) to get more and more water (current) through the pipe.

This relationship between the current (amount of water), the voltage difference (pressure difference), and the resistance of the bulb (the resistance of the pipe), is modeled by Ohm's Law as:

Voltage Difference = Current $*$ Resistance

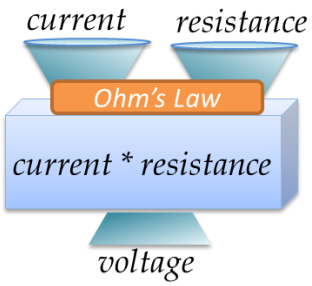
Or more succinctly:

$$V = I * R$$

Or in terms of our analogy:

Pressure difference = Amount of water through pipe $*$ Resistance of the pipe

Using our Visual Model, we can immediately see that the voltage is a function of two variables — namely, Current and Resistance. However, if we fixed the value of Resistance as in our example (i.e., we pick a certain light bulb), then Resistance becomes a constant input. Then, the relationship between Voltage and Current becomes linear.



This is a common technique when studying functions with multiple inputs. We fix all but one input, thereby making those inputs constant. The function now has only one variable input. We will use this strategy throughout this chapter to analyze various laws of physics that involve more than one input.

Ohm's Law is one of the most fundamental laws of electricity. All your electronic gadgets follow this fundamental law in one form or another. Isn't it amazing that we can analyze it using a simple model, with just one linear term (for a given resistance)?

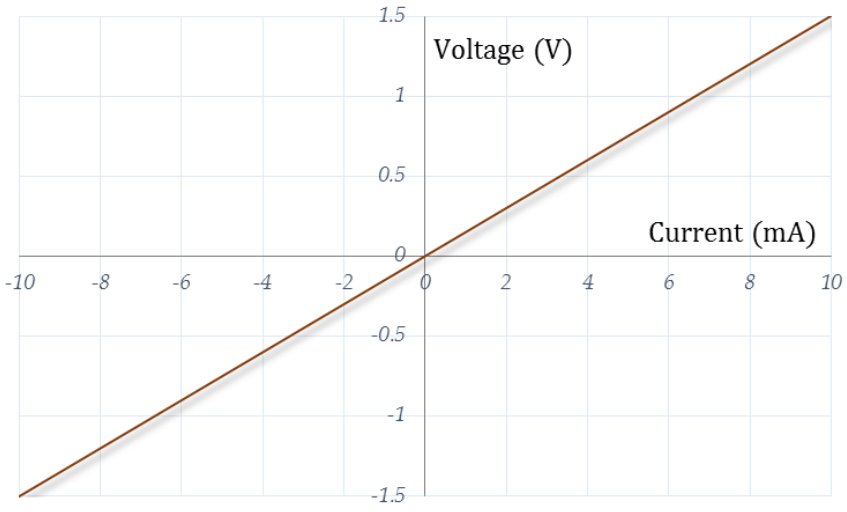
The computer code for Ohm's Law is given below. If you provide the `current` (in Amperes) and the `resistance` (in Ohms) as input, the function outputs the `voltage` (in Volts).

```
function getVoltage( current, resistance )
{
      voltage = current * resistance
      return voltage
}
```

```
volts = getVoltage( 3, 5 )  // input: 3 Amps, 5 Ohms
print( volts )              // output: 15 Volts
```

The graph of Voltage vs. Current through a resistor is given below. Rather unsurprisingly, the graph of this linear term is a straight line. The graph shows the voltage difference (output) across a given light bulb (i.e., a resistor) as we increase the current (input) through it. When we have a linear relationship like this, we say that Voltage is **directly**

proportional (or simply, **proportional**) to Current. Notice that voltage and current have a "sense" or "direction", so they can be negative as well.



Let's look at another law: Newton's **second law** of motion:

$$\text{Force} = \text{mass} * \text{acceleration}$$

Or, more succinctly,

$$F = ma$$

Again, Force depends on two input variables. However, for a given mass (for a given object with fixed mass), the relationship between force and acceleration is linear. If we want to double the acceleration, we have to double the Force applied to the object. Computer code to calculate the force for a given mass and acceleration is given below:

```
function getForce( mass, acceleration )
{
      force = mass * acceleration
      return force
}
```

```
force = getForce( 3, 5 )    // input: 3 kg, 5 m/s²
print( force )              // output: 15 Newtons
```

4.1.1.1 Reducing Non-Linear Models to Linear Models

A linear model is a very simple model to analyze and understand. Unfortunately, the nature is not always that simple. Many important physical relationships are not linear. Fortunately, there is a trick we can use to analyze non-linear models as linear models. We used this tactic when working with Ohm's Law in the previous lesson. Now, we are going to use the same trick on a more complicated relationship.

This time, we are going to look at the **ideal gas law**, which is a fundamental relationship in physics about a gas in a container. It is expressed as:

$$PV = kT$$

or

$$\text{Pressure} * \text{Volume} = k * \text{Temperature}$$

where k is a constant (which is equivalent $n*R$, where n is the number of moles of gas and R is the universal gas constant). Whenever a gas (or air) is in a container, whether it is in a cylinder of a car engine, or in your lungs, this law models how that gas behaves.

How should we look at a model like this? We always want to look at relationships between inputs and an output. There are three variables in this equation. Therefore, let's reorganize this equation in 3 ways, based on the output variable.

$$P = kT / V \quad \textbf{(1)}$$
$$V = kT / P \quad \textbf{(2)}$$
$$T = PV / k \quad \textbf{(3)}$$

In each of the above relationships, our output is expressed as a function of two variables (k is constant if we keep the number of gas molecules constant). However, none of the above are linear relationships, but we can use our favorite technique to analyze them as linear models.

Let's start with (1). If we make V our constant input, we end up with a linear relationship between P and T:

$$P = (k/V) * T \quad \textbf{(1A)}$$

Since both k and V are constant, we can write the relationship as follows, where $m = k/V$:

$$P = mT, \quad \textbf{(1B)}$$

where m is a constant.

What does this mean? If the volume V is constant (i.e., we do not change the volume of the container the gas is in), then, if we increase temperature T, the pressure P also rises proportionately. Whoa! We have a nice linear relationship. The slope m is k/V, which means, smaller the V, larger the m is. This means that if the gas is in a smaller container, when temperature T is increased, pressure P increases at a faster rate (compared to if the volume was larger). This makes physical sense. If the same number of gas molecules is in a smaller container, molecules are closer together. Thus, as we increase temperature, the vibrating molecules exert more pressure on the container walls. Do you see how much a linear model can teach us?

Now, let's take equation (2) and do a similar analysis. This time, to make this a linear relationship, we will make P a constant. That is, the gas will now be under a constant pressure. We begin with:

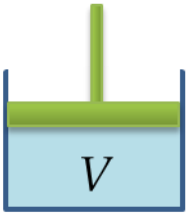
$$V = kT/P$$

By grouping the constants together, we get:

$$V = (k/P) * T \qquad \textbf{(2A)}$$

If we substituted constant m for k/P we get:

$$V = mT \qquad \textbf{(2B)}$$



Notice that this m is different from the m we used in (1B). What does this model tell us? If we do not change the pressure (and the number of gas molecules), when we increase the temperature T, volume V increases proportionately. For instance, the figure to the right shows a piston (green) of some weight applying pressure on a gas (blue) in a cylinder. As temperature rises, V increases raising the piston. In terms of physics, when we increase the temperature, molecules vibrate more energetically, forcing the volume to increase. Let's look at m as we did before. Since $m = k/P$, if we have lower pressure (a lighter piston), m is higher and the Volume increases at a faster rate as we increase the temperature, compared to a situation where the pressure is higher (a heavier piston).

We can do a similar analysis with (3):

$$T = PV / k$$

First, let's make pressure P constant. Grouping constants, we get:

$$T = (P \,/\, k) * V \qquad \textbf{(3A)}$$

This is the same relationship as (2A) between V and T. Similarly, if we make V constant, we get:

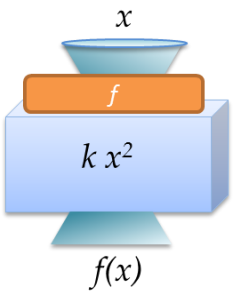
$$T = (V \,/\, k) * P \qquad \textbf{(3B)}$$

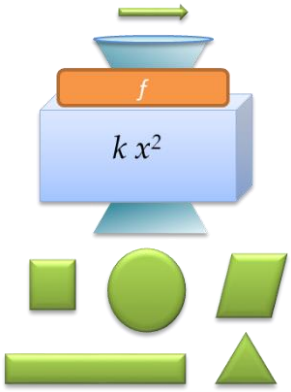
which is the same relationship as (1A) between P and T. Therefore, (3) is just a different way of expressing (1) and (2) – i.e., (3) is not independent from (1) and (2).

This example should clearly show you how we can make use of linear models to analyze more complicated non-linear relationships. You will find this technique quite helpful in analyzing many real-world models.

4.1.2 Quadratic Term

A relationship with a quadratic term takes the form $f(x) = kx^2$, where k is a constant and x is the input. The input is multiplied by itself, making the input more "powerful" in creating the output. As we discussed at the beginning of this chapter, this is the hallmark of power functions.

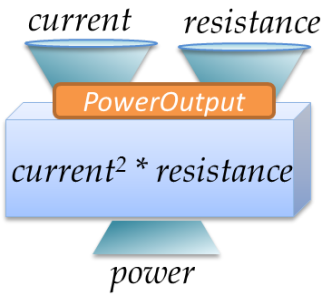




We can look at how this works using geometric objects, as shown in the Visual Model. If we input length x, at the output, we get objects represented by kx^2 (i.e., k*length^2) — i.e., objects with an area, like squares, circles, rectangles, triangles, etc. For instance, we can represent a square with k=1 (i.e., $1x^2$), an equilateral triangle with $k = ½$ (i.e., $½x^2$), and a circle with $k = \pi$ (i.e., πx^2). Therefore, the kx^2 term signifies area, if the input is treated as length.

Note that, even if the input has a sense (negative or positive), the output does not, because we square the input. That is, both positive and negative inputs map to the same positive output. Further, notice that $f(x) = kx^2$ has a root at zero because the output is zero when the input is zero.

As a real-world example of a relationship of the form $f(x) = kx^2$, let's examine the **electrical power** dissipated by a resistance (e.g., a lightbulb) due to current flowing through it.



$$\text{Power} = (\text{Current})^2 * \text{Resistance}$$

Again, notice that Power is a function of two variables. However, if the resistance is fixed (e.g., for a specific lightbulb), the relationship between Power and Current is quadratic. Computer code for this model is given below.

```
function getPower( current, resistance )
{
      power = current * current * resistance
      return power
}
```

```
watts = getPower( 3, 5 )    // input: 3 Amps, 5 Ohms
print( watts )              // output: 45 Watts
```

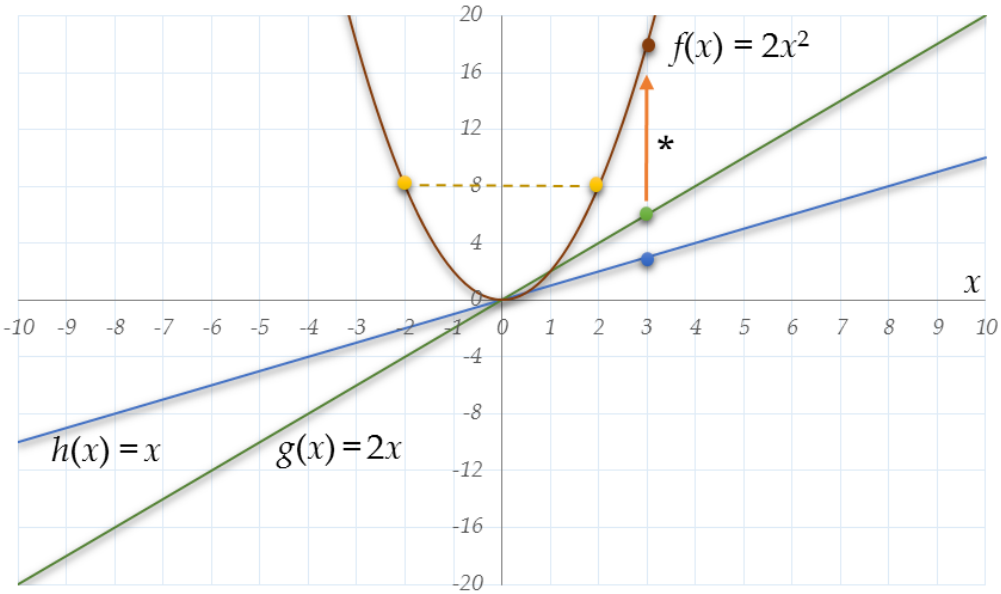
What's the significance of this law? Have you ever seen a blown fuse? How about a melted wire? Have you boiled water in an electric kettle or made toast with an electric toaster? All of this is made possible by this law.

When current flows through a resistor, the power generated is dissipated as heat. This could have both useful and disastrous effects in the real-world. If an excessive amount of current flows through a piece of wire, which has some resistance, that could melt the wire and cause an electrical fire. The same happens to a fuse, when it blows, but this time, protecting valuable equipment. On the other hand, the heat generated by a resistor can be used to heat up water (in an electric kettle), air (in a space heater), or make toast (in a toaster).

Here is the most interesting part: the power is proportional to the square of Current, which means that, if we double the current, the power output quadruples. That is why it is so easy to start electrical fires. Did you realize the role played by a power function here? It increased the "power" of the Current (input), allowing the input to have a bigger (more powerful) effect.

I want to draw your attention to one statement I made above: "the power *is proportional to* the square of Current". This means that the relationship between Power and Current^2, is linear. Notice that I said Current^2, not just Current. If you have a difficulty understanding this, replace, Current^2 with another variable like u. Then, Power = u * Resistance. This is

clearly a linear relationship. Variable substitution is another technique for looking at non-linear relationships as linear relationships, because it makes analysis simpler. However, do not forget that the underlying relationship between Power and Current is quadratic.



The graph of the quadratic function $f(x) = 2x^2$ is shown above, with the linear functions $h(x) = x$ and $g(x) = 2x$. Notice how we can obtain $f(x)$ by multiplying $h(x)$ by $g(x)$. For instance, at x=3, $h(x)$ is 3 and $g(x)$ is 6. By multiplying these two values together, we can find $f(6)$: 3∗6=18. This is shown by the **blue**, **green**, and **brown** dots on the graph. This shows how we can obtain a quadratic term as a product of two linear terms. Further, notice that $f(x)$, being a quadratic function, grows much faster than the linear functions $g(x)$ or $h(x)$. There is one other important thing, shown by the **yellow** dots on the graph. See how $f(x)$ has the same output when the input is 2 and -2. That is,

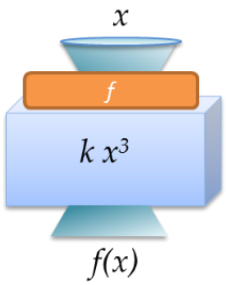
$$f(2) = f(-2) = 8$$

This shows that there are two inputs that produce the same output. That is, $f(x) = f(-x)$, a property we call ***even* symmetry** (symmetry about y-axis), because every power function with an *even* power has this property. As we will see shortly, this property becomes a real headache for us when we have to find the inverse of functions with this property.

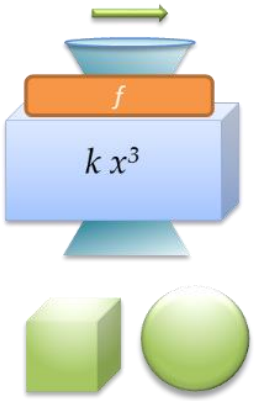
Power functions with even powers have *even symmetry*: $f(x) = f(-x)$

4.1.3 CUBIC TERM

A relationship with a cubic term takes the form $f(x) = kx^3$, where k is a constant and x is the input. As with all power functions, the input is made "more powerful" by multiplying itself repeatedly to raise it to the 3rd power.



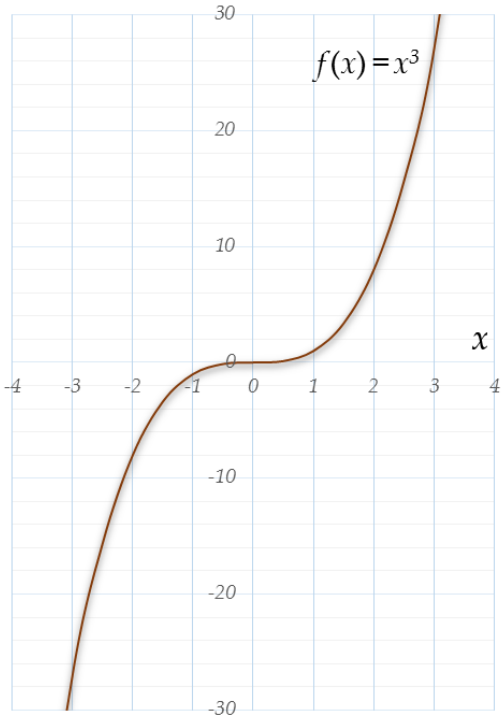
We can look at cubic terms as geometric objects, as shown by the Visual Model to the right. If we input some length x, at the output, we get objects that are represented by cubing that length — i.e., x^3 or length^3. These are objects with a volume, like cubes, spheres, prisms, etc. For instance, we can represent a cube with k =1 (i.e., $1x^3$), a triangular pyramid (tetrahedron) with $k=\sqrt{2}/12$ (i.e., $\sqrt{2}x^3/12$), and a sphere with $k = 4\pi/3$ (i.e., $4\pi x^3/3$). Therefore, kx^3 signifies volume, if the input is treated as a length.



If the input has a "sense" (negative or positive), the output will have the same sense too. That is, positive inputs produce positive output and negative inputs produce negative output. Therefore, if a physical model produces positive outputs, such models should accept only positive inputs.

```
function getVolumeOfSphere( radius )
{
      PI = 3.14159265
      volume = 4 / 3 * PI * radius * radius * radius
      return volume
}
```

Computer code for calculating the volume of a sphere of a given radius is shown above. You will most likely see a cubic model when you want to calculate the volume of some 3D object.



The graph of the $f(x) = x^3$ is shown in the figure. Do you see that every input value has a unique output value, unlike with the quadratic term? Because of this, we can find the inverse of cubic functions.

Notice that $f(x) = kx^3$ has a root at zero, because the output is zero when input is zero.

This graph is symmetric around the origin — i.e., if we rotate this graph by 180^0 around origin, it is unchanged. That is, $f(x) = -f(-x)$. Note that $-f(-x)$ means that we use an input with sense reversed and then reverse the sense of the output. This property is also true for the linear term we saw before. In fact, every power function with an *odd* power has this property. Therefore, this property is called ***odd* symmetry**.

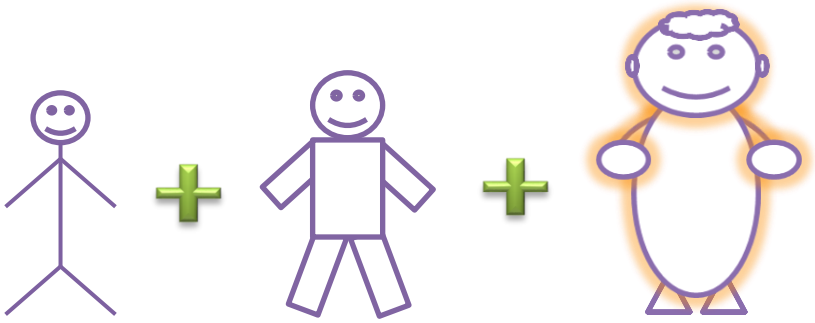
Power functions with odd powers have odd symmetry: $f(x) = -f(-x)$

4.2 Polynomial Family: How to Accumulate Different Powers

In the previous section, we met the members of the Power Function family. If you add two or more members of the power function family, also called terms, you get a **polynomial**. You can think of this as assembling a team of family members from the power function family, where each member is contributing a different power term to the output. For instance,

$$f(x) = 5x^3 + 7x^2 - 9x + 5$$

is a polynomial, with 4 terms. "But", you may object, "5 is not a member of the power function family". Trivially, it is, because x^0 is 1, and hence 5 can be written as $5x^0$. This is called the **constant term**, and if we express it as $f(x) = 5$, it becomes a constant function, which has a constant output, no matter what its input is. Can you see how different members of the power family contribute to create a polynomial?

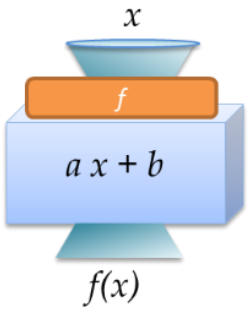


A polynomial: accumulation of different powers

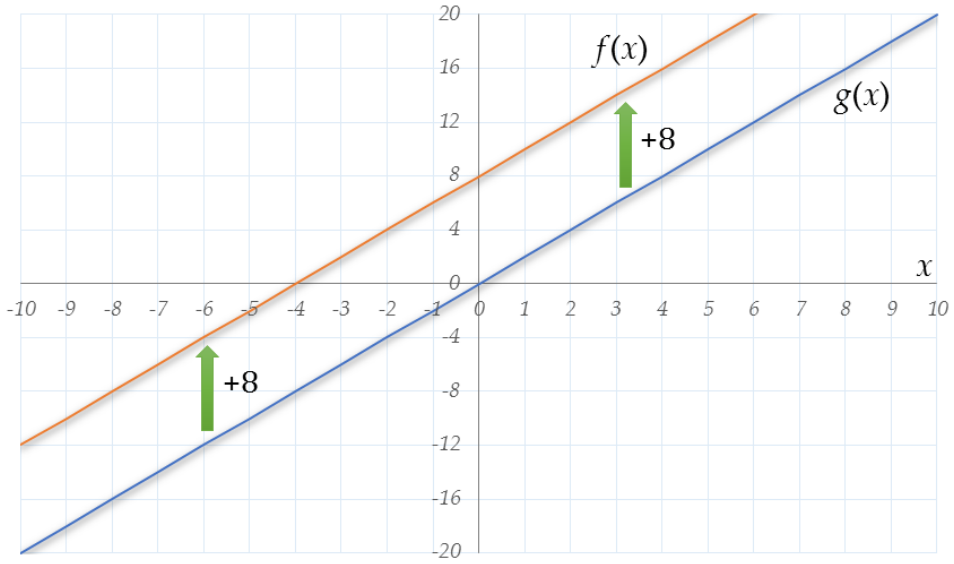
Since there are many different members of the Power Function family, there are many ways to combine these members with addition. Therefore, Polynomial Functions are a family as well. We will now take a look at the most common functions in this family.

4.2.1 Linear model (Linear Function)

A linear model consists of both a linear term and a constant term. It takes the form, $f(x) = ax + b$, where both a and b are constants.



For instance, $f(x) = 2x + 8$ is a linear function. The number 8 is the constant term. It does not change with the input. Therefore, it will always shift the output by a given offset (8, in this case). As an example, compare $f(x) = 2x + 8$ and $g(x) = 2x$. Function $g(x)$ is the same as $f(x)$, but without the constant term 8. The output of $f(x)$ is always 8 units more than the output of $g(x)$. As shown below, $f(x)$ is always 8 units above $g(x)$, which does not have any constant term. That's the effect of a constant term on any model (not just the linear model).



The linear model is quite common. For instance, if you want to convert from Celsius (^{0}C) to Fahrenheit (^{0}F), you would use the following linear model:

$$\text{Fahrenheit} = (9/5) * \text{Celsius} + 32$$

You should be able to immediately recognize this as a linear model, with a constant offset of 32. Why do we have an offset of 32? Remember, water freezes at 0 ^{0}C or 32 ^{0}F. Therefore, when the input is 0 ^{0}C, the output has to be 32 ^{0}F. The computer code for converting Celsius to Fahrenheit is given below.

```
function convertCelsiusToF( celsius )
{
      fahren = celsius * 9 / 5 + 32
      return fahren
}
```

```
fahren = convertCelsiusToF( 25 ) // input: 25 0C
print( fahren )                  // output: 77 0F
```

In the real-world, constant terms are often necessary when modeling relationships. For example, say you buy pencils, each at $1. Therefore, if you buy 10 pencils, you would have to pay $10. That's a variable cost, because the cost varies, linearly, with the number of pencils. However, you may have to pay a shipping charge of $5 regardless of the number of pencils you buy. That's a fixed cost, which is modeled using a constant term. So, your model becomes:

Total cost = \$1 ∗ *Number of Pencils* + \$5

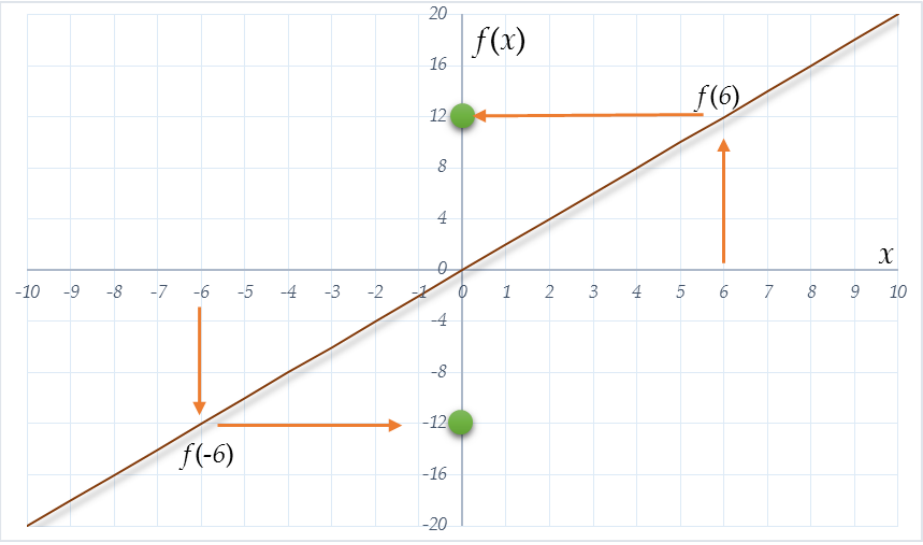
4.2.1.1 Significance of the Absence of a Constant Term

In a linear model, the absence of a constant term leads to a root at zero (you can generalize this observation to any polynomial). As we saw earlier, the constant term introduces a shift, or an "offset" or "bias". That is, in such a relationship, the magnitude of the output is different based on whether the input is negative or positive. For instance, the inputs -5 and +5, do not produce outputs with the same magnitude (hence "biased" towards either negative or positive inputs). For instance, on the above graph for $f(x) = 2x + 8$, $f(5)$ is 18 but $f(-5)$ is -2. However, on the graph for $g(x) = 2x$, $g(5)$ is 10 and $g(-5)$ is -10.

When there is no constant term, the relationship has no bias. Therefore, we call it symmetric about 0. Since $f(x) = ax$ has no constant term, this relationship is symmetric about 0. Say we input a value x and get an output y. If we input the negative of that value ($-x$), we get the negative of that output ($-y$). Notice that the magnitude of output does not change. *If we change the sign of the input, the sign of the output changes*, which we can express as $f(-x) = -f(x)$. That is,

if $f(\text{input}) = \text{output}$, then $f(-\text{input}) = -\text{output}$.

The following graph shows this fact for $f(6)$ and $f(-6)$.



What are the real-world implications of this symmetry? As an example, let's look at Newton's Second Law, $F=ma$. For the purposes of this example, let's consider east as our

positive direction. When determining force using $F=ma$, both the force (F) and acceleration (a) must be in the same direction. That is, if acceleration is in the eastward direction, the force is in the eastward direction too. What if we have a negative acceleration? Then the object is accelerating westwards. When that happens, from $F=ma$, the force (F) has to be negative too, since acceleration is negative. Thus, the force would be pushing the object westward. The magnitude of the acceleration is unchanged in both cases. Only the direction changes. In other words, this means that the relationship $F=ma$ holds if we apply it to an object moving eastwards or westwards.

The same reasoning applies to the Ohm's Law, where the current and voltage have a direction. Physical laws usually show this symmetry, and mathematically, in a linear model, you can see how it shows up as a root at zero — i.e., without a constant term or "bias". For physical laws, symmetry is a huge deal. It tells a fundamental property about the universe we live in. For instance, if Newton's Second Law did not have symmetry, we may have to do more work to go east compared to going west! Such a universe would be fundamentally different from the one we live in.

4.2.1.2 Linear Models of Multiple Inputs (Linear Combinations)

So far, we have looked at linear models of just one input. How about linear models (functions) of multiple inputs? Such functions are called **linear combinations**, because they build a "linear" function by "combining" multiple linear terms, each with a single input. Let's look at an example.

Let's say you want to make a special punch. You mix 2 cans of orange juice, 3 cans of apple juice, and 1 can of mango juice to make the punch. As you may have noticed, I am using some artistic license when it comes to food recipes, but you need to work with me here. The final product (the punch) is the result of a linear combination. In other words, the punch you made is a linear combination of orange, apple, and mango juice:

$2*\text{orange} + 3*\text{apple} + 1*\text{mango}$ [a linear combination]

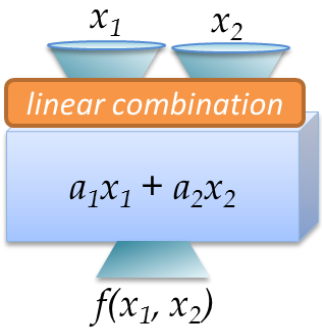
Given that you have to use the above three juice varieties, when making this punch, you were allowed to use only two operations: pick the number of cans from each variety, and then add them together to the punch bowl. Similarly, a linear combination allows only those two operations:

(1) Each input is multiplied by a scalar [pick the number of cans]
(2) Each term produced in (1) is added together [add to the punch bowl]

Those are the only two operations allowed in a linear combination (for a given number of inputs). For instance, multiplying one input by another is not allowed. Similarly, you cannot

exponentiate (raise the power of) an input, take the square root of an input, nor take the logarithm of an input. I think you get the point. Thus, a linear combination is a special (restricted) expression of multiple variables.

You can look at a linear combination as a sum of multiple linear terms, where each linear term has only one input variable. In fact, rule (1) above creates a linear term of one variable and rule (2) just adds those linear terms together. For instance, in our punch, 2∗orange is a linear function, and so is 3∗apple. More formally, we can write a linear combination of two variables x_1 and x_2 as:



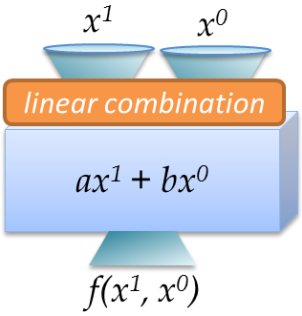
$$f(x_1, x_2) = a_1x_1 + a_2x_2$$

Notice that this is a function of two variables. Further, each term (e.g., a_1x_1) can be thought of as a linear term of one variable.

A linear combination is a sum of multiple linear terms

In essence, you are adding a bunch of input variables together, where each input variable is multiplied by a scalar. That's a linear combination. You can think of punch as a linear combination of juices. Similarly, you can think of salt water as a linear combination of salt and water, or air as a linear combination of Nitrogen, Oxygen, and Carbon dioxide.

If you think about it, the linear function $f(x) = ax + b$ can be viewed as a linear combination of two inputs: one variable input and one constant input, as shown with the Visual Model. In fact, we can generalize this to other polynomials, as we will see shortly.



Although, a linear combination looks quite trivial, as you will see in future chapters, linear combinations are quite versatile and fundamental. We will encounter linear combinations throughout this book in sections on polynomials, series, complex numbers, vectors, and matrices — all these are linear combinations of different inputs.

4.2.2 Quadratic Model

A quadratic function is nothing but a quadratic term plus a linear function. Therefore, it takes the form

$$f(x) = a\,x^2 + b\,x + c$$

The above general form has two roots, given by the famous quadratic formula. Note that, the roots can be complex (more on this in Section 4.2.4.2). We derive the quadratic formula by completing roots, which is factoring. That is, we can represent any quadratic model as a product of its factors. Recall that this is true for numbers:

$$6 = (2)(3)$$

Here, 2 and 3 are the factors of 6. The product of these factors leads to number 6. Similarly, the product of two factors can represent a quadratic model:

$$a\,x^2 + b\,x + c = (a_1 x + b_1)(a_2 x + b_2)$$

For example:

$$g(x) = x^2 + 2x - 3 = (x + 3)(x - 1)$$

Another example:

$$f(x) = 2x^2 - x - 3 = (2x - 3)(x + 1)$$

This means that $(2x-3)$ and $(x+1)$ are factors of $f(x)$. If we know the factors, we know the roots. Function $f(x)$ is zero when either of those factors is zero, because zero multiplied by anything is zero.

So,	$f(x) = 0$	when	$2x - 3 = 0$	OR	$x + 1 = 0$
That is,	$f(x) = 0$	when	$x = 3/2$	OR	$x = -1$

Now, here is an interesting intuition. Take a closer look at each factor. What kind of a model is that? For example, what kind of a model is $2x - 3$? What kind of a model is $x + 1$?

You guessed right! They are linear models (functions). So, what did we achieve by factoring the quadratic model? We decomposed it into a product of two linear models!

The quadratic model is a product of two linear models

This is a fascinating way to look at the quadratic model. Essentially, we can build a quadratic model by multiplying two linear models together. Conversely, if we have a quadratic model, we can break it up as a product of two linear models. That's not all. The linear models give us the roots of the quadratic model. Conversely, given the two roots, we can build up the two linear models and multiply them to get a quadratic model.

Take a moment to think about this relationship between factors, linear models, and the roots. You can see that the linear models are the fundamental building blocks (factors) of the quadratic model. Hence, the roots of the linear models become the roots of the quadratic model!

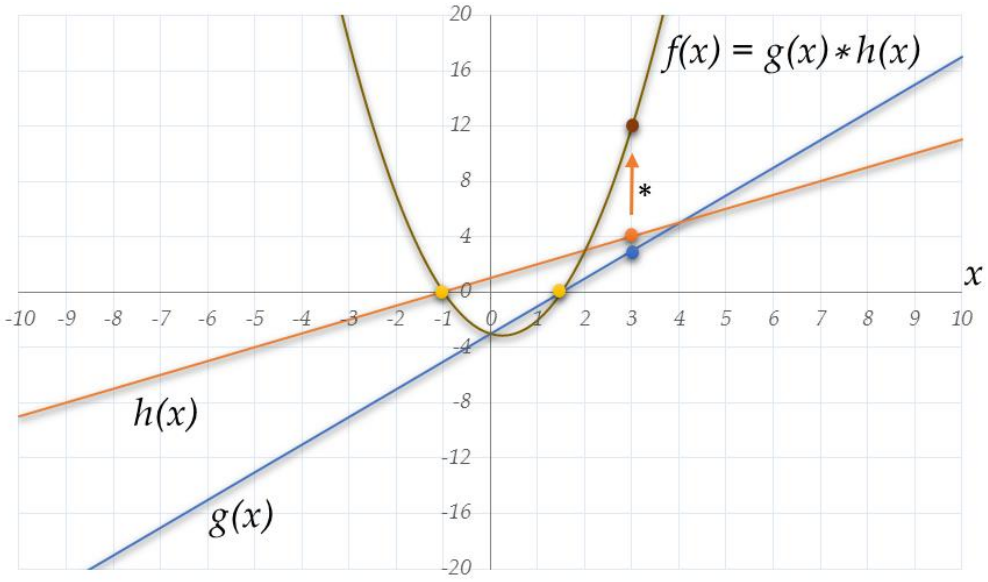
A quadratic function and its two linear factors have the same roots

Recall from Section 2.3.1 that roots represent solutions of an equation. In other words, if we can break up a quadratic model as a product of two linear models (linear factors), we can solve any quadratic equation! This is exactly what the famous quadratic formula does: it gives the two liner factors that yield roots (solutions).

The above is a general observation that holds true for any polynomial as we will see soon. This fact is shown graphically for the quadratic model $f(x)$ in the graph below, where $f(x) = 2x^2 - 5x - 3$. It has two factors, $(2x - 3)$ and $(x + 1)$. Let's name these factors $g(x) = (2x - 3)$, and $h(x) = (x + 1)$. Then, we can write $f(x)$ as:

$$f(x) = (2x - 3)(x + 1)$$
$$= g(x) * h(x)$$

The graphs of $f(x)$, $g(x)$, and $h(x)$ are given below.



The above graph shows how we can express a quadratic model as a product of two linear models. For any given input value x, we can find the output of f by multiplying the output of

g and output of h at the same input value x. For instance, at x=3, $g(x)$=3 (**blue** dot) and $h(x)$=4 (**orange** dot), and their product is 12 (**brown** dot), which is the output of $f(3)$, as shown on the graph. Please take a moment to understand this completely.

What's the relationship between the roots of these functions? Remember, the roots are the input values that produce an output value of zero. Roots are shown with the **yellow** dots on the graph. The linear models have roots at x=-1 and x=3/2, as you can see from the graph. Those are also the *roots* of the quadratic function and the *solutions* to the equation $f(x) = 0$. Again, please take a moment to study the graph and understand this completely.

4.2.3 Cubic Model

In the last section, we saw that a quadratic model is a quadratic term plus a linear model. Similarly, a cubic model is a cubic term plus a quadratic model. I think you can see the pattern now.

4.2.4 Generalization: Polynomial Functions

Linear, quadratic, and cubic models we met so far belong to the same polynomial family. For instance, the cubic model is a polynomial of degree 3, because the highest order (power) term is x^3. Similarly, a quadratic model is a polynomial of degree 2.

We can extend this to higher powers of input as well. For instance, a 4th degree (quartic) polynomial is a x^4 term (in the form ax^4) plus a cubic model (cubic polynomial).

You can look at a polynomial as a sum (a series) of terms, which are members of the power function family. For instance, a cubic polynomial can be viewed as a sum (a series) of a cubic term, a quadratic term, a linear term, and a constant term, as shown below:

$$f(x) = \_\_\_\, x^3 + \_\_\_\, x^2 + \_\_\_\, x^1 + \_\_\_\, x^0$$

A blank space ___ represents a place to plug in a constant coefficient. More formally, we can represent this as:

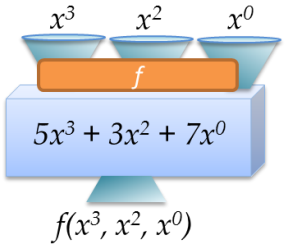
$$f(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0x^0$$

where a_3, a_2, a_1, a_0 are constants. For instance, $5x^3 + 2x^2 - 9x + 2$ is a third-degree polynomial function.

Notice that polynomials are a family of functions. Their individual family members include functions like linear, quadratic, and cubic functions, among others. When we want to study such functions in general, we can put all of them under the umbrella of a generic polynomial.

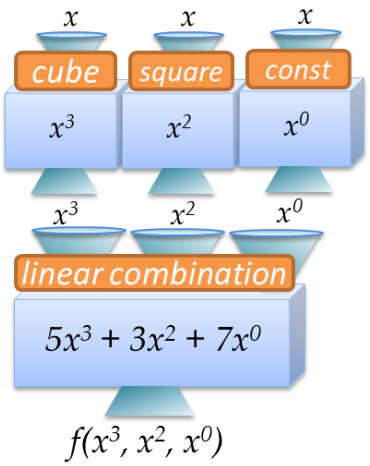
4.2.4.1 Polynomials as Linear Combinations

Here is a different way to state what we saw above: polynomials are a *linear combination of power functions.* For instance, take the power functions x^3, x^2, and x^0. You can look at each of them as a different input — e.g., apple, mango, and orange juice, as in our punch example. Now, if we combined these three inputs in different proportions, we can get a polynomial like:



$$5x^3 + 3x^2 + 7$$

Notice that this is not a function of x; rather, x^3, x^2, and x^0 (power functions) are the three inputs of the linear combination, as shown with the Visual Model. If this is confusing, let $u=x^3$, $v=x^2$, and $w=x^0$, and we can write the above polynomial as:



$$f(u, v, w) = 5u + 3v + 7w$$

Here, we multiplied each input (power function) by a scalar, and added them together. The Visual Model shows how a polynomial is composed out of power functions using a linear combination. That is, multiple power functions are feeding in to the linear combination. We can write this polynomial as a function of 3 power functions:

$$f(x^3, x^2, x^0) = 5x^3 + 3x^2 + 7$$

A polynomial is a linear combination of power functions

4.2.4.2 The Fundamental Theorem of Algebra

In Section 4.2.2, we saw how a quadratic model can be viewed as the product of two linear models. This means that the function has two roots. The Fundamental Theorem of Algebra generalizes this to any polynomial. It says that a polynomial of degree n has n roots. The roots may not be real (i.e., roots are complex) in some cases, as we will see in Section 6.3. Some roots may be repeated (i.e., not distinct).

A polynomial of degree n has n roots (real/complex)

To summarize, a polynomial model of degree n can be expressed as a product of n linear functions. Each linear function is called a factor of the polynomial. *The root of each linear factor is also a root of the polynomial.* The same roots are also solutions of the equation $f(x) = 0$, where $f(x)$ is the polynomial.

The root of each linear factor is also a root of the polynomial

In other words, solving a polynomial equation of the form $f(x) = 0$ boils down to finding its linear models (factors), and finding their roots. A polynomial is a product of linear models. Nothing more. Nothing less. That's it. It's similar to factoring a number like 6. Number 6 has two primary factors, 3 and 2, so 3×2 leads to 6.

If $f(x)$ is a polynomial of degree n, $f(x)=0$ has n solutions (real/complex)

Isn't it fascinating that any polynomial, which is a *sum* (a series) of power functions, can be expressed as a *product* of linear models (factors)?

Looking at a polynomial as a product of linear models (factors), helps us understand what a polynomial is. At the same time, perhaps even more importantly, it helps us understand what a polynomial is *not*. If you cannot express any model as a product of linear models, it's not a polynomial. Let's look at a few examples.

Take a model as straightforward as $f(x) = 1/x$. It cannot be expressed as a product of linear models. Therefore, we cannot model such a function with a polynomial. Similarly, you cannot model $f(x) = \sqrt{x}$ or any other fractional or negative powers of x, as a product of linear models. You cannot express any ratio of polynomials as a product of linear models — e.g., $f(x) = (x-1)/(x+1)$. You cannot model $f(x)=2^x$ as a product of linear models. Thus, none of these are polynomial functions.

The above examples show that polynomials are not the only game in town. In fact, they can't be because they have a big limitation. A polynomial can represent only products of linear models. Therefore, we need other models to capture relationships that polynomials cannot model. In the following sections, you will meet such models that cannot be expressed as a product of linear models.

4.3 Exponential Family

Lose weight in just 30 days! No pills! No dieting!! You can eat anything you want, even a whole pizza every day! Why spend 3 months dieting when you can get the results you want in just 1 month! Results guaranteed, or your money back! All this is for one easy payment of $19.95. Are you ready for the challenge? Have your credit card ready and call the number on your screen!

Okay, I hear that you are ready pay? Great! You just made a great investment in your own health. Ready to hear the details of the plan? Here it is. You just have to do pushups for just 30 days. That's it. Didn't I say results are guaranteed?

Here is the exact plan. The first day, you have to do only 2 pushups. Can you believe how easy this is? The second day, you just have to double that. Just 4 pushups! Easy as that! Every day, you double the number of pushups you did the previous day. If you continue doing this for just one month, you are guaranteed to lose weight, or your money back. Period.

Now that you paid $19.95, let's figure out what your schedule is going to look like:

On the 1st day, you start with 2 pushups.

On the 2nd day, you multiply 2 by 2. i.e., $2 \times 2 = 4$

On the 3rd day, you multiply 4 by 2. i.e., $2 \times 2 \times 2 = 8$

On the 4th day, you multiply 8 by 2. i.e., $2 \times 2 \times 2 \times 2 = 16$

So, far so good. I think you get the pattern now. Mathematically, this is raising the number 2 to some power, where the power is the number of the day since start. So, the number of pushups you have to do on any day is given by:

calculatePushups (day) = 2^{day}

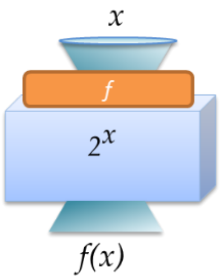
where calculatePushups is the function that takes the day as its input and outputs the number of pushups you have to do on that day. If you like more succinct notation, we can write the same function as:

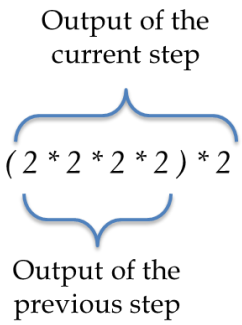
$f(x) = 2^x$, where x is the day.

This is when you realize the problem with the plan. On the 10^{th} day, you have to 2^{10} pushups — or 1024 pushups. Let's say that somehow you do that, but on the 20^{th} day you would have to d 2^{20} pushups. Whoa! That's more than 1 million. When you get to the final day, you would have to do 2^{30}, or more than 1 billion, pushups!

If you continue with this plan, not only are you guaranteed to lose weight, but you are also guaranteed to lose your limbs! Time to call a lawyer.

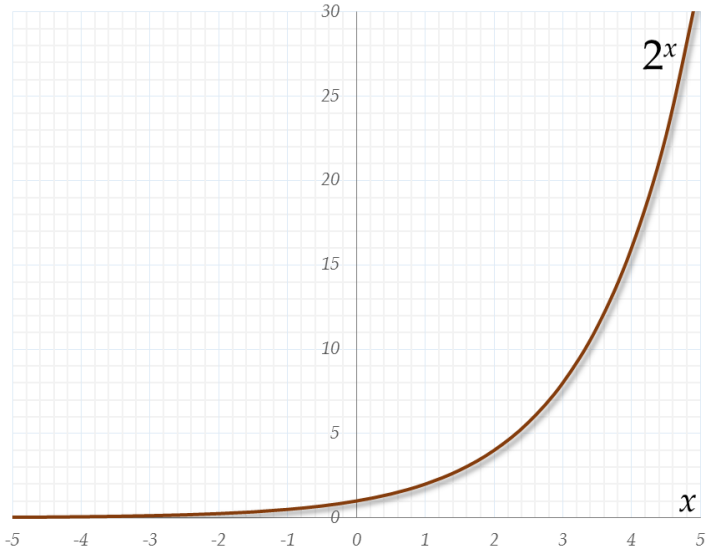
How did you end up like this given that you just started with 2 pushups on day 1? It's all thanks to the model we used: the **exponential** model (function). In an exponential model, at each step, the output is calculated by multiplying the *output of the previous step* by some constant amount. In the above example, the constant amount is 2. This value is called the **base**. At each step, we multiply the output of the previous step by the base, which is 2 in the above example. This is illustrated below. You may also recognize this repeated multiplication as exponentiation, as we saw in Section 3.4.





In concise terms, this exponential model can be represented as $f(x) = b^x$, where b is the constant base. Notice that this is very different from $f(x) = x^2$, $f(x) = x^3$, etc., which are just power functions. In a power function, a variable "input" is raised to a constant power like 2,

3, etc. That is, the "input" is multiplied a given number of times by itself. In an exponential model, a constant number (a base) is raised to a variable power, which is given by the input. Because of this, at each step, the *output* of the previous step (output of the previous input value) is multiplied by a constant. In our pushup example, the output doubled every day. Since the output kept doubling, the function grew rapidly (i.e., exponentially). Because of this, in everyday language, people tend to refer to any rapid growth as exponential growth. That is not always correct. In math, exponential growth has a precise definition: it's a function where the output of the previous step is multiplied by a constant multiplier to get the output of the current step. This produces rapid growth as shown by the graph of $f(x) = 2^x$, shown below.



Here is another way to look at this rapid growth in output. The exponential model greatly amplifies the difference in input. For instance, if we have $g(x) = x^2$, the ratio between $g(10)$ and $g(5)$ is $100/25 = 4$. Now, if you look at the same ratio for the same input interval with $f(x) = 2^x$, we get $f(10)/f(5) = 1024/32$, or 32. In addition, a difference of mere 5 in input became a difference of $1024 - 32 = 992$ for output.

Although they have very different growth rates, x^2 and 2^x have one thing in common: both x^2 and 2^x are constructed using the exponentiation operation. Hence, they are related to the general two-input power function (exponentiation operator) we saw in Section 3.4. For instance, x^2 = power(x, 2), and 2^x = power(2, x). Notice what is constant and what is variable in each model.

Computer code for calculatePushups is given below to calculate the number of pushups needed for a given day. This code has two new features. First, it uses an if-else statement. It

is pretty straightforward. It says that "**if** day is equal to 1", then the "result is 2"; "**else**, the result is …". Note that we use the equality operator, ==, to test for equality.

```
function calculatePushups( day )
{
      if (day == 1)
            result = 2
      else
            result = 2 * calculatePushups( day - 1 )

      return result
}
```

```
pushups = calculatePushups( 20 )
print( pushups )
```

The 2nd new feature is found in the statement

```
result = 2 * calculatePushups( day – 1 )
```

This means that the number of pushups for 'day' is 2 times the number of pushups on the previous day, which is "day–1". That is, the number of pushups for day 4 is two times the number of pushups needed on day 3. Note that the above expression, `calculatePushups (day – 1)` is a function evaluation (function call); make sure that you do not confuse that with the function definition at the top of the code.

In case you are interested, when a function calls itself (as shown above), it is called **recursion**. Here, we calculate the output for the current step using the output of the previous step. This recursive process goes on until the input (day) is 1. Then, the '`if`' statement becomes true and the function returns 2, thereby terminating the recursive process. Many mathematical definitions show this relationship and hence naturally lead to recursion. If you are a programmer, you should know how to convert this to an iterative statement (a `for` loop), but I will not do it here, because recursion is a fundamental programing structure we need to understand to express mathematical relationships.

Exponential models have many real-world uses. One very common use is calculating compound interest (on your savings, or on a loan). Say you deposit $100 in the bank today. The bank is going to pay you 2% interest on the balance each year. What will be your balance after 10 years? Let's work it out for a few years first:

After 1 year, the balance is $100 * 1.02 = $102.00 **(1)**

After 2 years, the balance is $102 * 1.02 = $104.04 **(2)**

This is crucial to understand. The bank pays interest on the entire balance, which includes your principal $100 you deposited initially, and the interest you earned the previous year (i.e., the total output of the previous step). So, (2) can be written as:

$$100 * 1.02 * 1.02$$

After 3 years, the balance is:

$$100 * 1.02 * 1.02 * 1.02$$

Now, you can see a pattern emerging. Every year, we multiply the previous year's balance (output) by 1.02. This is very similar to the pushup example, where we multiplied the previous day's pushups by 2. After n years, the balance will be:

$$\$100 * (1.02)^n$$

We can model this as follows, where 100 is the principal (initial deposit):

$$\text{Balance } (n) = 100 * 1.02^n$$

When n=10, we get the balance after 10 years, which is equal to $100*(1.02)^{10} = \$121.90$.

Since we multiply the previous balance (output) by a constant (1.02) each year, this is **exponential growth**. Computer code for calculating the balance after a certain number of years is given below. For instance, to calculate the balance for the above situation, we can call calculateBalance(100, 0.02, 10).

```
function calculateBalance( principal, rate, year )
{
      if (year == 0)
            balance = principal
      else
            balance = (1.0 + rate) * calculateBalance( year - 1 )

      return balance
}
```

```
balance = calculateBalance( 100, 0.02, 10 )
print( balance )
```

4.3.1 THE (NATURAL) EXPONENTIAL FUNCTION

In our pushup example, $f(x) = 2^x$, we had an exponential function with base 2. In the bank-balance example where $f(x)=100*(1.02)^x$, the base was 1.02. An exponential function can have any base. However, is there a base that is more useful than any other? There is and it's the irrational number 2.7182…, which is represented by the letter e, just as 3.14159… is represented by the letter π. Why is e more useful than other bases? To understand this, we have to look at the *slope* or the *rate of growth* of an exponential function.

The rate of growth (slope) of an exponential model depends on the value of the base and the input value (x). As an example, the middle row of the following table shows the *output* value of three exponential functions at x=4: 2^x, e^x, and 3^x — that is, 2^4, e^4, and 3^4. The last row shows the *slope* of the same functions at x=4. As you can see, the slop of e^x at x=4 is the same as output value at x=4 (i.e., e^4). For e^x, this is true for all other values of x (not just x=4). You can see this relationship does not hold for bases 2 (2^x) or 3 (3^x). For base 2, the slope is always smaller than the output value, as shown in the graph below. For base 3, the slope is always greater than the output value. That suggests that there is a base between 2 and 3, where the slope is always equal to the output value. That base is e, where e is 2.7182…

	2^x	e^x	3^x
output at x=4	16.00	54.60	81.00
slope at x=4	11.09	54.60	88.99

What's the significance of the above property? To fully answer that, we need calculus (a mathematician would say, e^x is the eigenfunction of the derivative operator, which, to most people, sounds like "blah blah blah"). For our purposes, it just means that if we ever want to find the slope of e^x at any point, we don't have to sweat at all — we can just use the output value of e^x at that point. This makes e^x special. Therefore, we call it **the natural exponential function**.

Although e^x has special properties, for the purpose of understanding, it models a similar function to our pushup routine. In our pushup routine, each day, we did 2 times the pushups we did the previous day. If we use e^x instead, we would be doing 2.7182 times the pushups we did the previous day. That's it. Instead of doubling, we multiply by a little over 2.7. So, intuitively, just look at e^x as a slightly modified pushup routine (I will leave it up to your imagination to figure out how to do a fractional pushup). Can you modify the computer code of the calculatePushups function, so that each day we would do 2.7182 times the pushups we did on the previous day?

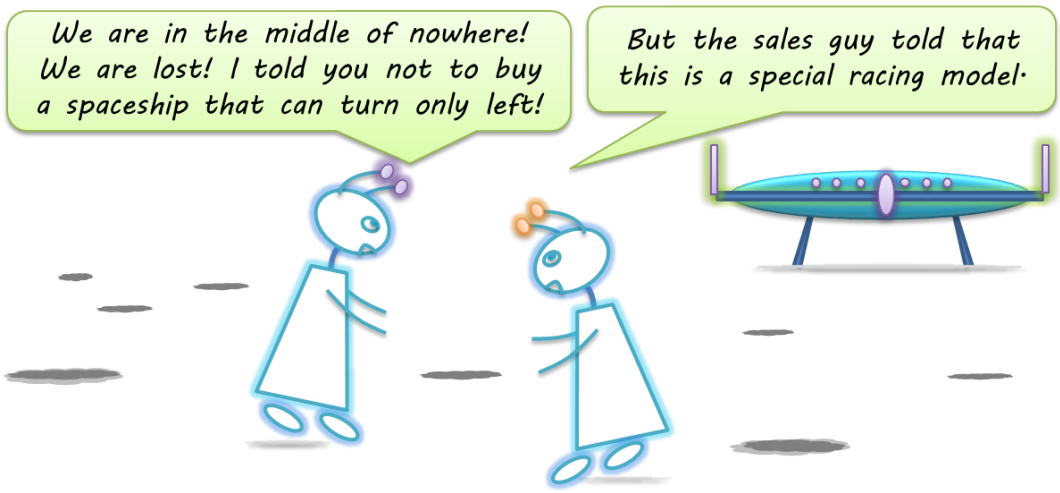
4.4 The Inverse Functions of The Families We Met

Recall that, so far, we have met 3 function families: (1) Power functions, (2) Polynomial functions, and (3) Exponential functions. In this section, we are going to look at their inverse functions, if they exist.

Again, recall that a function and its inverse can undo each other's effect. If we feed the output of a function into its inverse, we get the original input back (as the output of the inverse function). In other words, a function helps us get the output value y for a given input value x. However, if we know the output value, y, to get back to the original input, x, we can use the inverse function.

Remember, in Section 2.2, we saw that if you know the inverse function of $f(x)$, you know the solution to an equation of the form $f(x) = b$. That is one of the reasons why we are so interested in inverse functions — they help us solve equations.

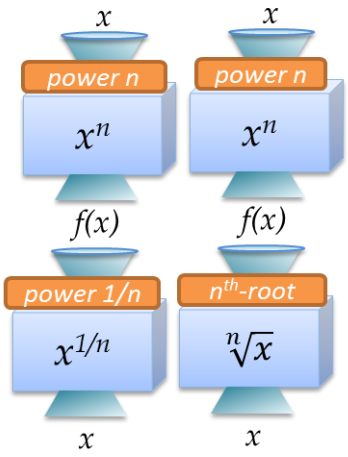
Therefore, it is always useful to look at a function and its inverse together. For instance, if you can turn your head left, you'd better know how to turn it right as well, if you ever want to face straight again.



4.4.1 Inverse of Power Functions

Recall that a function of the form $f(x) = x^n$ is a power function, where n is a positive integer *constant*. For instance, $f(x) = x^2$ and $f(x) = x^3$ are power functions. We saw how these types of power functions are used in building a polynomial.

In general, if $f(x) = x^n$, the inverse function $f^{-1}(x)$ is given by the **n^{th} root** of x, as shown by the Visual Model.



$$f^{-1}(x) = x^{1/n} \qquad [\ n^{th} \text{ root of } x\ ]$$

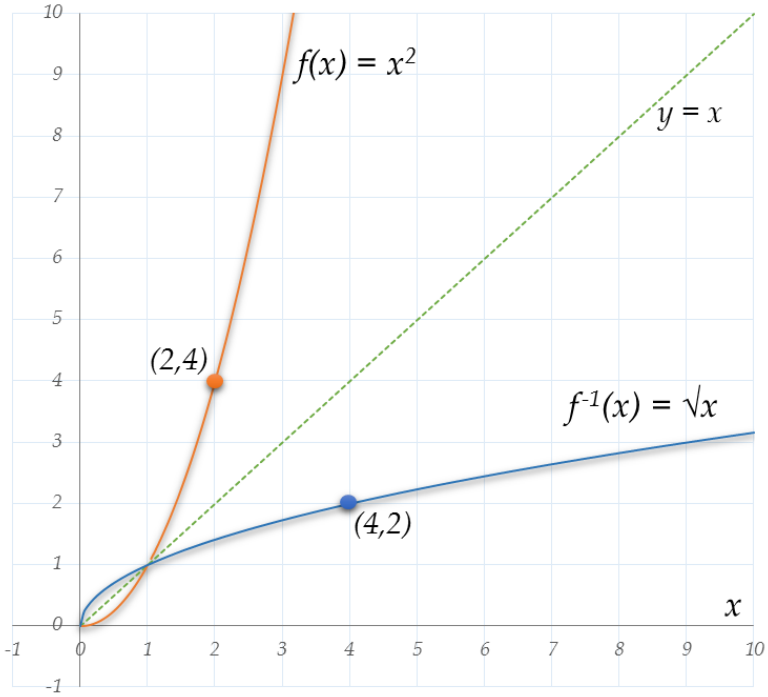
$$= \sqrt[n]{x} \qquad [\ n^{th} \text{ root of } x\ ]$$

There is one important caveat. We learned that if a function produces the same output value for different inputs, that function does not have an inverse, unless we restrict the domain of that function's input values. For instance, when n is even, the power functions produce the same output for different inputs. Therefore, we need to restrict the input domain. With that in mind, let's look at the inverse of different power functions we encountered.

Let's start with n=1. If $f(x) = x$, its inverse is itself, because $f(x) = x$ is the identity function, where input and output are the same. Therefore, $f^{-1}(x) = x$.

When n=2, we have $f(x) = x^2$. Its inverse, $f^{-1}(x) = x^{1/2} = \sqrt{x}$. This is the **square root** function.

The graph of $f^{-1}(x) = \sqrt{x}$ is given below. Notice that $f^{-1}(x) = \sqrt{x}$ is parabolic, just as $f(x) = x^2$ and f^{-1} is defined for only positive values of x.



Graphically, an inverse function is a reflection of the original function around $y = x$ line (i.e., identity function). Can you imagine why? Recall the definition of inverse. If function f produces output y for input x, then the inverse function should accept y as input and produce x as output. For instance, if function f produces 4 as output for an input of 2 (orange dot), the inverse function should produce 2 as the output for an input of 4 (blue dot). Notice that the perpendicular distance from point (4, 2) to the line $y = x$ is the same as the distance from the point (2, 4) to the line $y = x$. This symmetry is present between *any* function and its inverse, provided that the inverse exists.

Notice that $f(x) = x^2$ produces the same output for two input values. For instance, when x=2 or x=-2, the output is 4. However, the inverse function, $f^{-1}(x) = \sqrt{x}$ gives only the positive value. We can represent the negative value using the inverse function $f^{-1}(x) = -\sqrt{x}$. We can combine both functions and write:

$$f^{-1}(x) = \pm \sqrt{x} \qquad \textbf{(1)}$$

It is important to notice that (1) represents two functions at once:

$$f^{-1}(x) = +\sqrt{x} \quad \text{and} \quad f^{-1}(x) = -\sqrt{x}$$

Therefore, you can think of (1) as a *mathematical relation* capturing two functions. A mathematical relation, unlike a function, can have multiple outputs for the same input. To summarize, to find the inverse of $f(x) = x^2$ for all the real values of x, we need two inverse functions.

The other functions of the power family and their inverse functions behave the same way. When n=3, we have $f(x) = x^3$. Its inverse, $f^{-1}(x) = x^{1/3}$, is the **cube root** . The graph of $f(x) = x^3$ rises faster than that of $f(x) = x^2$. Therefore, their inverse functions have the opposite relationship: $x^{1/3}$ rises *slower* than $x^{1/2}$. Since $f(x) = x^3$ produces a unique output for each input, its inverse exists for all values of x. For instance, the cube-root of 8 is 2, and cube-root of -8 is -2.

Do you know why $\sqrt{2}$ is called the square *root* of 2? What does this have to do with any root (zero) of a function? Well, say we write $f(x) = x^2$. Let's represent the output value of $f(x)$ with b. Then, we have $x^2 = b$. To solve this equation, we have to find the *roots* (zeros) of function $f(x) = x^2 - b$. Recall that roots are the solutions to the equation $f(x) = 0$. Since $f(x)$ is a 2nd degree polynomial function, it has two roots: one at $+\sqrt{b}$ and the other at $-\sqrt{b}$. Therefore, $\sqrt{b}$ and $-\sqrt{b}$ are the roots (zeros) of the function $f(x) = x^2 - b$. Similarly, a cube-root is a root of the function $f(x) = x^3 - b$.

As a real-world example of a square root, let's look at the relationship between current, resistance and power as we did before. We know that

$$\text{Power} = (\text{Current})^2 * \text{Resistance}$$

Therefore, if we want to find Current, given Power consumption and Resistance, we can express Current as:

$$\text{Current} = (\text{Power}/\text{Resistance})^{1/2}$$

Thus, Current is given by the square root of the ratio between Power and Resistance.

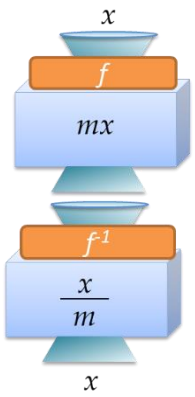
Since area has a quadratic relationship to length, square-roots naturally occur in problems where we know the area of an object and we want to find the lengths of its sides. Similarly, since volume has a cubic relationship to length, cube-roots occur naturally in problems where we know the volume of an object and want to find the lengths of its sides.

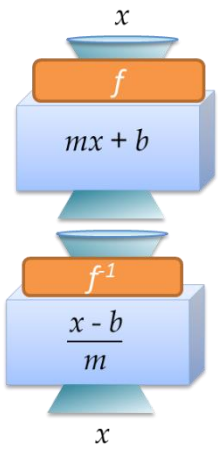
4.4.2 Inverse of Polynomials

In this section, we are going to explore the inverse of polynomial functions. We start with the simplest polynomial model: the linear model.

4.4.2.1 Inverse of the Linear Model

Our Visual Model shows the inverse function, f^{-1}, of the linear model $f(x) = mx$, where m is a constant and x is the input. As you can see, f^{-1} is also linear. The only difference is in the coefficient of multiplication (or slope of the line). That is, if $f(x) = mx$, then $f^{-1}(x) = (1/m)\,x$, or $f^{-1}(x) = x/m$.





If we have the general linear function $y = m\,x + b$, to find its inverse, we just solve for x as:

$$x = (y - b)/m$$

The right-hand side of the equation is a function of y, so we can write:

$$f^{-1}(y) = (y - b)/m$$

If we use x as the traditional input variable, we get:

$$f^{-1}(x) = (x - b)/m \qquad \textbf{(1)}$$

or

$$f^{-1}(x) = x/m - b/m \qquad \textbf{(2)}$$

Note that (2) is also a linear function, with a slope of $1/m$ and an offset of $-b/m$. This is an important observation. We don't need a different type of function to reverse the effects (undo) a linear model. This is not the case with most of the functions we will meet in this section.

A linear model can undo (invert) another linear model

As we saw before, the inverse of a function provides the general solution to the equation $f(x) = b$. For instance, if $f(x) = 5x + 3$, its inverse, $f^{-1}(x)$, is given by $(x - 3)/5$. So, if we have the equation $f(x) = 10$, the solution would be given by

$$f^{-1}(10) = (10 - 3)/5$$

$$= 7/5$$

To verify the answer, we can input 7/5 to function f, giving us $f(7/5) = 7*5/5 + 3 = 10$.

4.4.2.2 Inverse of Polynomials

Most polynomial functions don't have inverse functions, because they produce the same output for multiple inputs. However, we can find the inverse functions for some polynomials when we can restrict the domain of input values so that the function does not produce the same output more than once within the given domain.

To find the inverse function of $f(x)$, we use a variable, say y, to represent its output and then solve for x (because x is the output of the inverse function). For instance, assume we have:

$$f(x) = x^2 - x - 2$$

From here, let's use variable y to represent the output of $f(x)$ and solve for x.

$$y = x^2 - x - 2$$

or

$$0 = x^2 - x - 2 - y \qquad \textbf{(A)}$$

Notice that (A) is a quadratic equation of x. How do you solve for x in an equation like this? The easiest way is to use the **quadratic formula**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Using the quadratic formula, we can find the solutions to equation (A). Notice that, from equation (A), we get $a = 1$, $b = -1$, and $c = (-2 - y)$. If we plug these into the quadratic formula, we get:

$$x = \frac{1 \pm \sqrt{1^2 - 4.1.(-2 - y)}}{2}$$

$$x = ½ \pm ½ \sqrt{(1 + 8 + 4y)}$$

$$x = ½ \pm ½ \sqrt{(9 + 4y)}$$

As expected, we get two roots. Thus, we can write *two* inverse functions:

$$f^{-1}(y) = ½ + ½ \sqrt{(9 + 4y)} \qquad \textbf{(1)}$$

or

$$\underline{f}^{-1}(y) = ½ - ½ \sqrt{(9 + 4y)} \qquad \textbf{(2)}$$

Notice that function name $\underline{f}$ is underlined in equation (2) to distinguish it from equation (1). Also, notice that we expressed f^{-1} as a function of input variable y but we could have used any input variable name.

Say we wanted to find the input value that makes an output of 4. To do that have to find $f^{-1}(4)$ and $f^{-1}(4)$. Plugging 4 into equations (1) and (2) gives us:

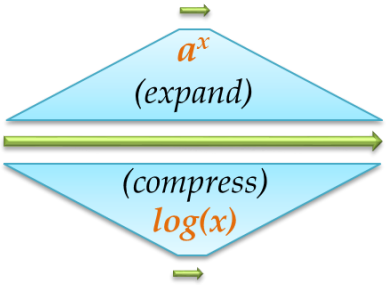
$$f^{-1}(4) = 3 \qquad \text{and} \qquad f^{-1}(4) = -2$$

If we check our answers using the original function, we see that $f(3) = 9 - 3 - 2 = 4$ and $f(-2) = 4 + 2 - 2 = 4$.

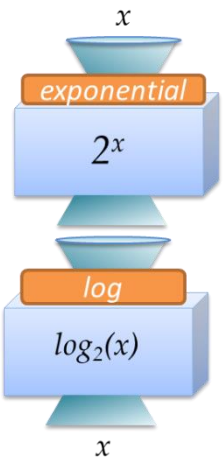
Notice that both equations (1) and (2) have real coefficients if and only if $(9 + 4y) \geq 0$. This arises from the fact that the output of $f(x)$ is always greater than $-9/4$. Therefore, we see that inverse functions with real coefficients are defined for a limited number of y values.

You can approach finding the inverse of any quadratic function the same way as we saw above. For higher degree polynomials, the inverse may not exist or could be hard to find. Therefore, as we discussed in Section 2.3.2, when we need to solve polynomial models of higher degree, we need to resort to other methods to find roots.

4.4.3 Inverse of Exponential Functions: Logarithmic Model



With an exponential model, we saw that the output grows really fast when we increase the input. The **logarithmic** model is the inverse of the exponential model. As a result, its output grows really slowly, when we increase the input. If you understood the inverse operation, this should not come as any surprise. The logarithmic model has to "undo" all the "expansion" that the exponential model did, to get back to the original input. Thus, you can think of the logarithmic operation as performing "compression" of input.



The Visual Model shows how the logarithmic function (of base 2) can undo the exponential function 2^x. As an example, for an input domain of 5 to 10, the exponential function 2^x produces outputs in the range of 32 to 1024. In contrast, for inputs between 32 to 1024, the output of $\log_2(x)$ is between just 5 and 10. Thus, logarithmic functions can "compress" a wide input domain to a very small output range.

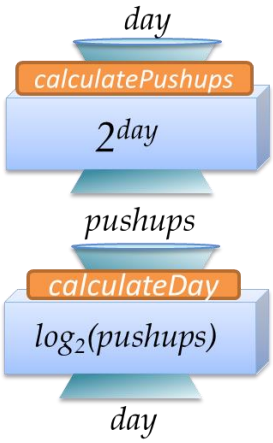
Let's look at a concrete example. Our calculatePushups function gave the number of pushups you need to do for a given day. In contrast, the logarithmic model would give us the number of days it takes to get to a given number of pushups. Such a logarithmic model can be defined as:

calculateDay(pushups) = $\log_2$(pushups)

The above will output the day at which we do a given number of pushups, if we were doubling the number of pushups each day. More concisely, we can represent it as:

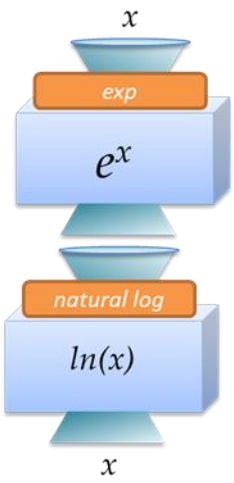
$f(x) = \log_2(x)$, where x is the number of pushups.

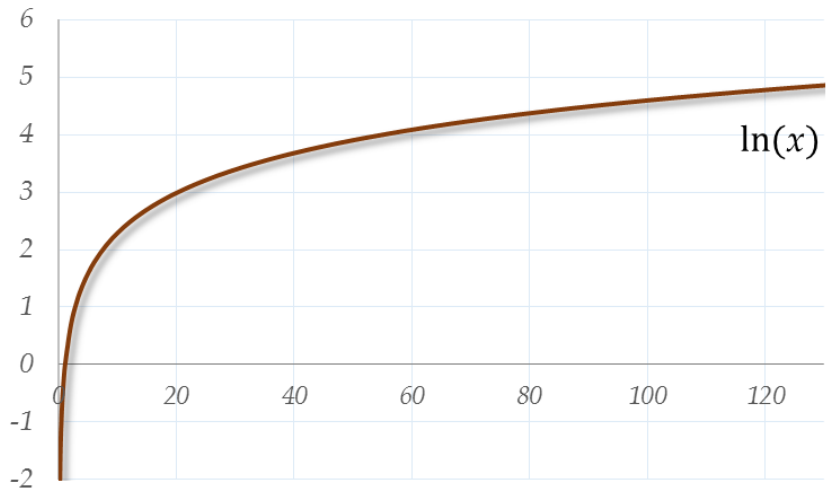
For instance, if we use 1024 as the input to the above logarithmic function, the output will be 10, which means that on the 10th day, you will have to do 1024 pushups, if we started with 2 pushups on day 1 and kept on doubling.



The Visual Model of our pushup routine should further clarify why the logarithmic and exponential models are inverse of each other. If we input a "day" to the exponential model, after sending its output through the logarithmic model, we get the same "day" back. As an example, if we did pushups for 10 days, on the 10th day, we will do 1024 pushups. Now, if we use 1024 as our input to the above logarithmic function, calculateDay, the output will be 10, getting us back to where we started.

Note that, in our figure, both the exponent and the logarithm are of base 2. However, the inverse relationship applies for any base. As an example, the Visual Model shows the same inverse relationship with base e. We call a logarithm with base e as "ln" or ***n***atural ***l***ogarithm.





You may have seen graphs with log axes — i.e., one or both axes are logarithmic. Do you know why we need them? It is tied to the "compression" property of the log function. As a practical example, let's take cell division. Cells replicate by repeated division.

At timestep 0, we start with 1 cell ["mother" of all cells]
At timestep 1, we have 1∗2 = 2 cells [1 cell divided into 2]
At timestep 2, we have 2∗2 = 4 cells [both cells divided into two, producing 4]
At timestep 3, we have 4∗2 = 8 cells [4 cells divided into two, producing 8]

Now, you know the drill. We have seen this many a time. This is describing an exponential function with a base of 2. At each step, each cell creates an output (another cell). You can visualize this as all cells at a given step doubling for the next step. That is, the output of each step is double that of the previous step. That's the salient feature of an exponential model.

We can represent the number of cells as a function of timestep:

$$\text{numberOfCells}(\text{timestep}) = 2^{\text{timestep}}$$

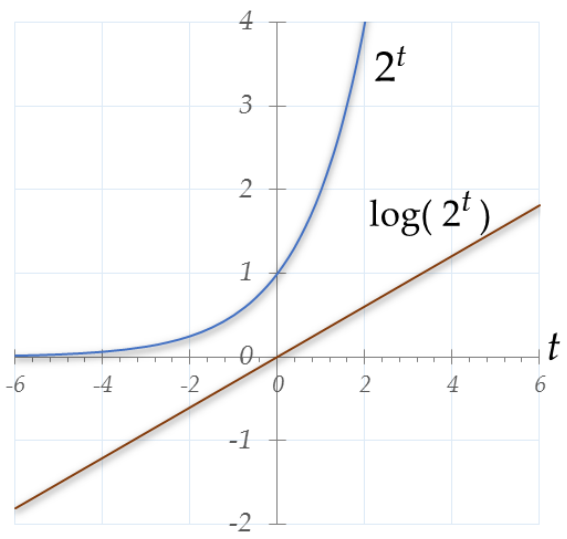
Or more succinctly, using single letters as variable names:

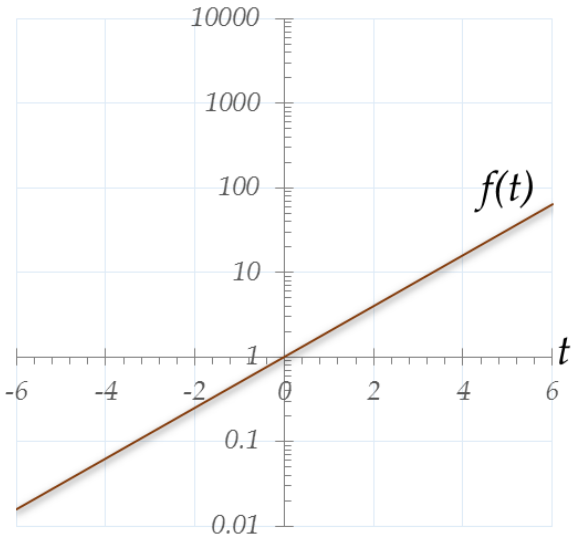
$$f(t) = 2^t$$

where t is the timestep. Remember, t starts at 0. If we plot the timestep, t, on the horizontal axis and the number of cells, $f(t)$, on the vertical axis, we get a graph similar to the one shown below on the left. Since cell-division is an exponential model, $f(t)$ grows rapidly. If we want to show 30 steps, on paper, we may have to buy our own paper mill!

How do you show an exponential relationship like this for a large input domain? Easy. Instead of plotting $f(t)$ on the vertical axis, we plot the *log of* $f(t)$ — i.e., $\log_{10}(f(t))$. That is, we send the output of $f(t)$ through a log function (with a base of 10) and plot that result. If we do that, the resulting line $\log(2^t)$ is shown on the left graph below (note that when we omit the base, it is assumed to be 10). This clearly shows the "compression" property of the logarithmic model, which compresses an exponential range to a linear range. By the way, did you notice that $\log(f(t))$ is function composition?

The graph of $f(t)$ is shown on the right as well, but with the vertical axis labeled differently. There, the vertical axis is usually called a "log axis". For this graph, as well for the previous one, we send $f(t)$ through a log function to get $\log(f(t))$ and plot the resulting value. However, to read out the values of $f(t)$ directly, the vertical axis is labeled with the *original* output of $f(t)$, instead of $\log(f(t))$. For instance, when the $\log(f(t))$ is 2 (left graph), $f(t)$ is 100 (right graph). Either way, the relationship between $\log(f(t))$ and t is linear, because the relationship between $f(t)$ and t is exponential. You will frequently see graphs with log axes in practice, because exponential relationships are common in the real world.





A practical use of a logarithmic scale is the **Richter scale** used to express the magnitude of an earthquake. Can you guess why? It's because, the magnitude of an earthquake can vary from small to very large. Some earthquakes are minor tremors you can hardly feel, whereas others flatten buildings and cause tsunamis. What's the best way to express such a wide range? Send it through a log function (in this case, with a base of 10), and use that output. Therefore, if the Richter scale identifies a magnitude 7 earthquake and a magnitude 8 earthquake, the latter is 10 times more powerful than the former. Similarly, **decibels**, denoted by dB, is another such metric used to express the logarithm of a ratio, instead of the plain ratio, because the range of the ratio can be really large.

We know that inverse functions are used to solve equations of the form $f(x) = b$, where b is a constant. We know that, $\ln(x)$ and e^x are inverse functions of each other. If we have $f(x) = e^{2x}$, and the equation $f(x) = 5$, we would get

$$e^{2x} = 5$$

This is an exponential model. Therefore, we can apply the inverse function of e^x, to both sides of the equation:

$\ln(e^{2x}) = \ln(5)$ [take log of both sides]

$2x = \ln(5)$ [ln(5) is a constant]

$x = \ln(5) / 2$

Similarly, if we have $f(x) = \ln(5x)$, and we have to solve the equation $f(x) = 2$, we could write:

$$\ln(5x) = 2$$

Applying the inverse function of $\ln(x)$ to both sides of the equation produces the following. Notice that you can think of e^x as **exp**(x) to visualize the application of the exponential function.

$e^{\ln(5x)} = e^2$ [exp(ln(5x)) = exp(2)]

$5x = e^2$ [e^2 is a constant]

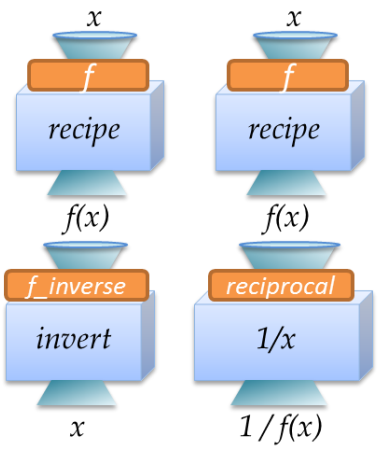
$x = e^2 / 5$

The value of inverse functions in solving equations of the form $f(x) = b$ should be clear to you by now.

4.5 Reciprocal Functions of the Families We Met

In the real-world, we often find relationships between input and output where the output decreases proportionately when the input is increased. Similarly, when we decrease the input, the output increases proportionately. We call this type of a relationship a **reciprocal** relationship. In this section, we are going to explore the reciprocal relationships of the 3 function families we have met so far.

Before we begin, there is a huge pitfall in naming you should be aware of. If we increase the input of a function and the output decreases proportionately, we also say that the output is **inversely proportional** to the input. However, an output being "*inversely proportional*" to an input is not the same as an "*inverse function*". They are very different concepts. The two Visual Models should make the difference clear as night and day. The appearance of word "inverse" in "inversely proportional" is quite unfortunate. Whenever, you should see *1/function*, the word that should immediately come to your mind is "***reciprocal***".



$f^{-1} \neq 1/f$, in general

Here's a good way to remember the relationship between a function and its reciprocal: if you multiply any function by its reciprocal, you should get 1 as the output:

$$f(x) * 1/f(x) = 1$$

This also means that, if the reciprocal of $g(x)$ is $f(x)$, then the reciprocal of $f(x)$ is $g(x)$, because $f(x) \,.\, g(x) = 1$. Therefore, $f(x)$ and $g(x)$ are ***reciprocals of each other***.

Notice that the same relationship is not true in general for an inverse of a function, f^{-1}. In general:

$$f(x) * f^{-1}(x) \neq 1$$

4.5.1 Reciprocal of Power Functions

4.5.1.1 Reciprocal of The Linear Term

The simplest power function we studied was $f(x) = x$, which has one linear term. The reciprocal of this, $g(x)$, is:

$$g(x) = 1 / f(x)$$

$$= 1 / x$$

Therefore, the simplest reciprocal model is $g(x) = 1/x$. We can verify this by using:

$$g(x) * f(x) = 1, \qquad \textbf{(1)}$$

which means that the function multiplied by its reciprocal is always 1. Note that $1/x$, is also known as the **multiplicative inverse** of x. This is another name used for the reciprocal. Again, it has nothing to do with inverse functions.

Here is another pitfall to be careful of: $1/x$ can be also written as x^{-1}, where x is raised to the power of negative 1. Never confuse this with $f^{-1}(x)$, which is inverse of f (think of $f^{-1} = f^{\text{inverse}}$). To avoid the overloaded use of word "inverse", we will continue to refer to $1/x$ as the reciprocal of x. In Section 4.4.1, we saw that the inverse of $f(x) = x$ is $f(x)$ itself, not $g(x) = 1/x$. This shows that the inverse of a function is very different from its reciprocal.

The more general form of (1) can be written as:

$$f(x) * g(x) = k \qquad \textbf{(2)}$$

where k is a constant. When k=1, we get (1). If $f(x)$ is increased by some factor, then $g(x)$ has to be decreased by the same factor to produce constant k as the output.

Why is the reciprocal model so important in practice? Well, as we will see soon, inverse proportionality is very common in the real world. For instance, take Ohm's Law,

$$\text{Voltage} = \text{Current} * \text{Resistance} \qquad \textbf{(3)}$$

If Voltage is constant, (3) becomes

$$\text{Current} * \text{Resistance} = k$$

This is a reciprocal relationship similar to (2). Another way to see this is to reorganize (3) as follows:

Current = Voltage / Resistance **(4)**

Equation (4) means that Current is *directly proportional* to Voltage but *inversely proportional* to Resistance. That is, if Resistance is increased by a factor of 10, Current is reduced by a factor of 10. This is a typical reciprocal relationship.

Let's take another physical law: the Ideal Gas Law we have met before. It says:

Pressure $*$ Volume $= k *$ Temperature

where k is a constant.

If Temperature is constant, we get a typical reciprocal relationship:

Pressure $*$ Volume $= k$ **(5)**

or

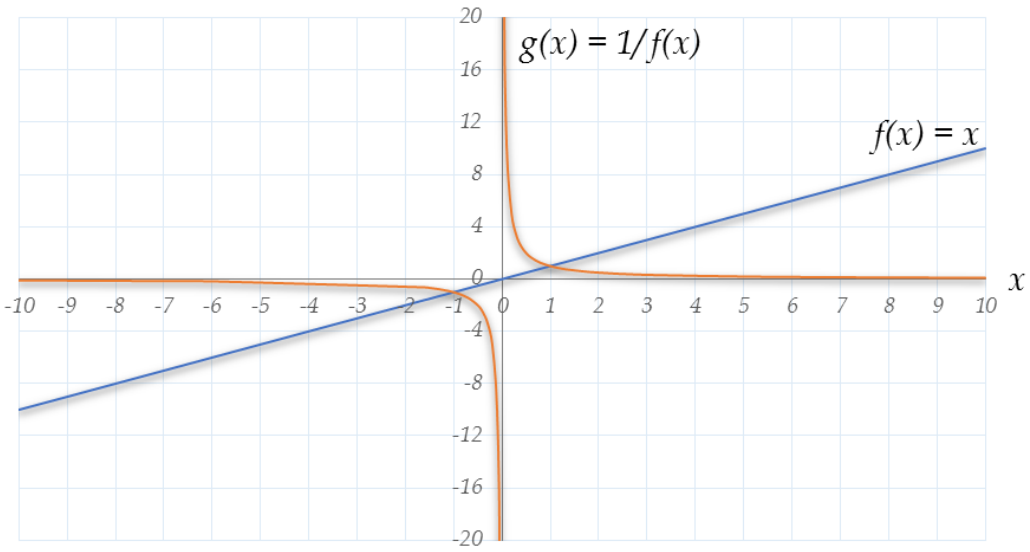
Pressure $= k$ / Volume **(6)**

For instance, if we double Volume, Pressure will be halved. Equation (5) is known as **Boyle's Law**, which is a special case of the Ideal Gas Law.

The graphs of $f(x) = x$ and its reciprocal model $g(x) = 1/x$ are given below. Notice how the orange curve for $g(x)$ goes to positive infinity as x approaches zero from the positive direction. This is because when we divide 1 by a very small positive value, we get a very large positive value. However, $g(x)$ goes to negative infinity as x approaches zero from the negative direction, because we are now dividing 1 by a very small negative value.

The following graph represents (6) when k=1. If x is Volume, then $g(x)$ represents Pressure. As expected, when Volume is increased, Pressure drops, and when Volume approaches zero, Pressure increases rapidly, approaching infinity.

Similarly, the following graph can represent (4), when Voltage is 1. If x represents Resistance, $g(x)$ represents Current. As Resistance increases, Current approaches zero. When Resistance decreases, Current increases, approaching infinity when Resistance approaches 0.



So far, we have looked at the function $f(x) = x$, which is also known as the identify function, because it simply outputs the input it receives. Let's look at the reciprocal functions of more general linear models.

If $f(x) = mx$, then its reciprocal is $1/mx$

If $f(x) = mx + b$, then its reciprocal is $1/(mx + b)$

The reciprocal of a linear model is a not a linear model.

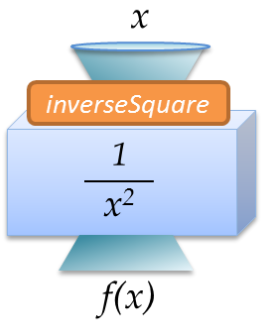
The reciprocal of a linear function is not linear

4.5.1.2 Inverse Square Model

In the previous section, we looked at the reciprocal of a linear term. Similarly, we can have the reciprocal of a quadratic term. For instance, if $f(x) = x^2$, then its reciprocal model is $g(x) = 1/x^2$.

The reciprocal function $g(x) = 1/x^2$ is known, rather unfortunately, as the "***inverse square***" function. As noted at the beginning of this chapter, this has nothing to do with inverse functions. It's a reciprocal relationship.

In the previous section, we looked at reciprocal models where the output is "inversely proportional" to the input. When output is inversely proportional to the *square of the input*, we get an inverse square model. That is, if we double the input, the output becomes a quarter of what it used to be.



The most famous relationship that shows inverse square behavior is Newton's **Gravitational Law**:

$$\text{Force} = G * \text{mass}_1 * \text{mass}_2 / \text{distance}^2$$

This law says that the gravitational force between two objects is *proportional* to the product of their masses and *inversely proportional* to the square of the distance between them. G is a constant. Force is a function of three input variables, as shown in the computer code below. If we fix the masses of the objects (i.e., for two given objects of constant mass), the Force is inversely proportional to the square of the distance between them. What does that mean? If we double the distance, the gravitational force between the objects becomes a quarter of what it used to be.

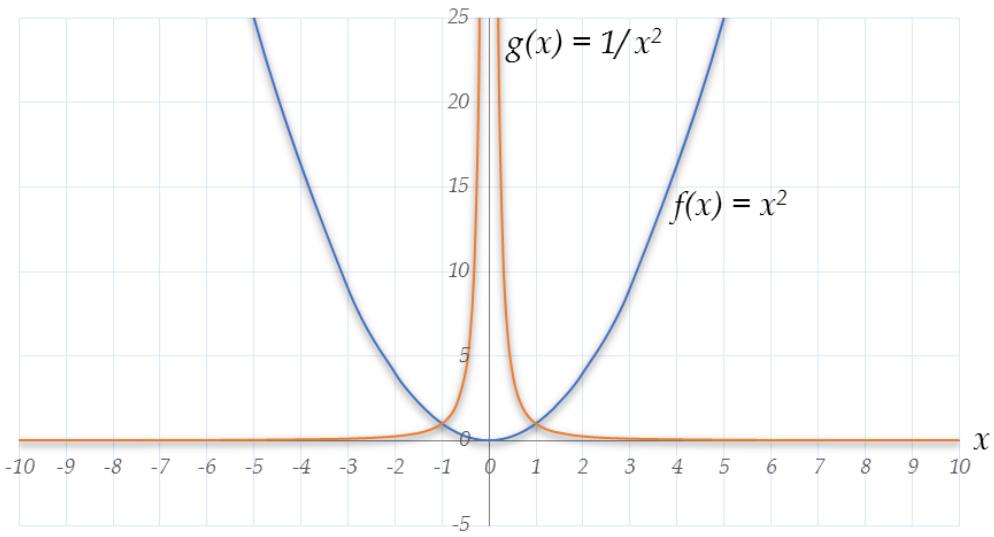
```
function gravitationalForce( mass1, mass2, distance )
{
      G = 6.67408e-11
      force = G * mass1 * mass2 / (distance * distance)

      return force
}
```

Another famous physical law that follows the inverse square model is **Coulomb's Law**:

$$F = k * q_1 * q_2 / d^2$$

where q_1 and q_2 are two electrical charges, k is a constant, and d is the distance between them.

The graph of $f(x) = x^2$ and its reciprocal model $g(x) = 1/x^2$ are shown below. If we compare this to the gravitational model with fixed masses of 1 unit, x represents the distance between the two objects and $g(x)$ represents the gravitational force between them. As you can see, as the distance between the two objects starts increasing, the gravitational force between them decreases rapidly. This is the reason why the gravitational force becomes so weak with increasing distance.



Again, recall that the inverse of $f(x) = x^2$ is $f^{-1}(x) = \sqrt{x}$, which is very different from the reciprocal of $f(x)$, which is $1/x^2$.

4.5.2 Reciprocals of Polynomials

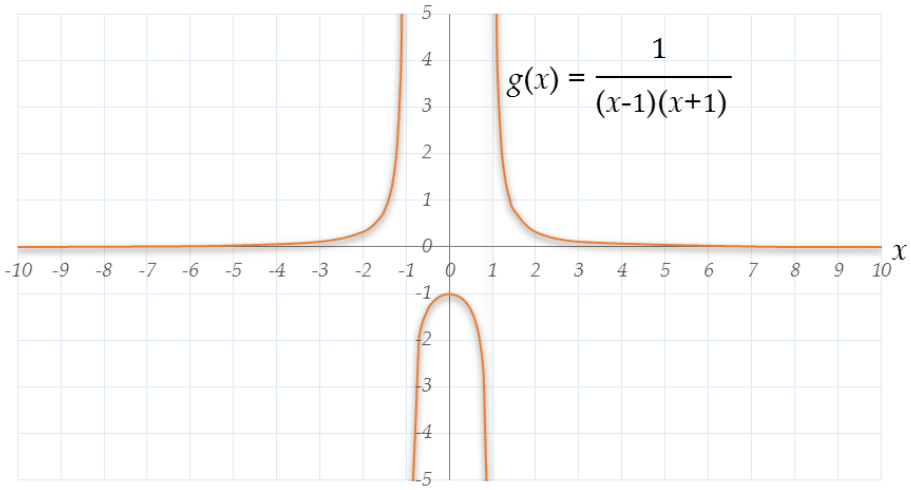
Just as we can find the reciprocal of a general linear model, we can find the reciprocal of quadratic, cubic, and other polynomial functions. For instance, if $f(x) = x^2 - 1$, we can write its reciprocal, $g(x)$, as:

$$g(x) = 1 / (x^2 - 1)$$

Since we can factorize $f(x)$ as $f(x) = (x - 1)(x + 1)$, we can also write the above as:

$$g(x) = 1 / (x - 1)(x + 1)$$

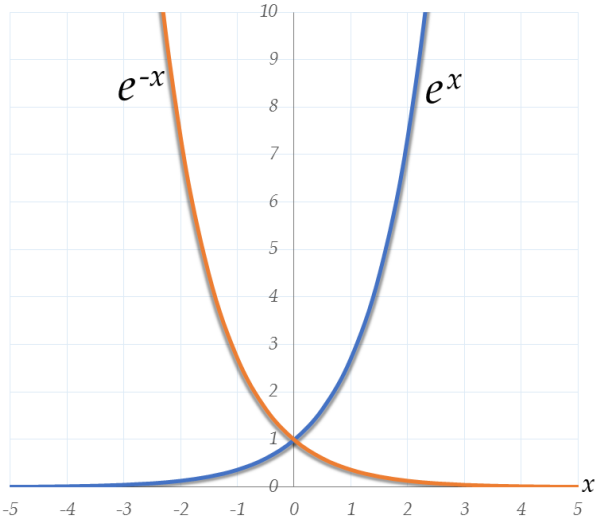
The graph of $g(x)$ is given below. As you can see from its graph, when $g(x)$ approaches 1 or -1 (i.e., the roots of $f(x)$), the output becomes infinite. The number of such inputs is given by the number of linear factors, or the number of roots, of the polynomial $f(x)$. From the fundamental theorem of algebra, we know that an n^{th} degree polynomial should have n linear factors or n roots, including repeated roots and complex roots. You will encounter these types of reciprocal models in areas like control systems.



4.5.3 Reciprocal of Exponential Models (Exponential Decay)

The reciprocal of exponential functions is quite common. For instance, the reciprocal of 2^x is 2^{-x} (or, $1/2^x$) because the product of the two functions produces 1. As an example, the graph of e^{-x} is given below, with e^x shown for reference. As you can see, at any given value of x, the product of e^x and e^{-x} is 1, and hence, both lines cross at (0, 1). That is:

$$e^x * e^{-x} = 1$$



The above property makes the functions e^{-x} and e^x reciprocals of each other. When x is positive, e^x is called an exponential growth model and its reciprocal, e^{-x}, is called an **exponential decay** model. Notice that e^{-x} is *not* the inverse of e^x. The inverse of e^x is the natural logarithm, $\ln(x)$.

As an example, let's say we had a pushup routine where we do only half the number of pushups we did the previous day. This is an exponential decay model. If you start with 1024 pushups, the next day you have to do only 512. By the 10th day, you are down to only 2 pushups: an ideal formula to *gain* weight!

If you consider the above to be a facetious example, a really important phenomenon also follows the same pattern: **radioactive decay**. If you have a lump of radioactive material, it emits radioactive particles. After a given number of years (say, 1 year), its radioactivity will

be only half of what it used to be. After another year, it will be half of that (i.e., a quarter of the original amount). The time period, 1 year in this case, is called "**half-life**" — the time it takes the radioactivity to reach half of its original amount. The "half-life" of our decaying-pushup example was just one day. For radioactive materials, it could be thousands of years.

This is the fundamental principle behind "carbon dating". No, it's not another online dating site; it's a method for determining the age of a material. We can easily explain carbon dating with our decaying-pushup example.

Let's say you and your sister start doing pushups — starting with 1024 the first day! However, you halve the number of pushups you do each day, while your sister does not. Let's say that on the N^{th} day, your sister is doing 8 times the number of pushups you're doing on the same day. Which day is this? Or, in other words, what is the value of N? Let's see. 1024 divided by 8 is 128. So, your sister did 1024 pushups as always while you did only 128. How many days did it take you to get to 128 pushups from 1024? Let's start at 1024 and keep on dividing by 2: 1024, 512, 256, 128. That is, on the 4^{th} day, you are doing 128 pushups. In other words, after 3 days, your sister was doing 8 times the number of pushups as you did.

Now, let's say that there are two elements that behave just like you and your sister. One element does not decay (your sister) while the other decays exponentially (as you did). We know the half-life of the decaying element (just like the half-life of pushup routine was just 1 day). Now say, we measure the ratio of radioactivity between these two elements in a sample and find out that the ratio is 8, just as in our pushup example. What does this mean? We know exactly it has taken three half-lives to get to a ratio of 8. If the half-life was 1000 years, we know that it has been 3000 years since these two elements were equal. That's the basis of carbon dating. One form of Carbon, Carbon-14 decays (just as you did) while the other form of Carbon, Carbon-12, stays the same. Therefore, by determining the ratio of Carbon-14 to Carbon-12 in an organic sample, you can determine its age. Our silly pushup example was not that silly, after all, was it?

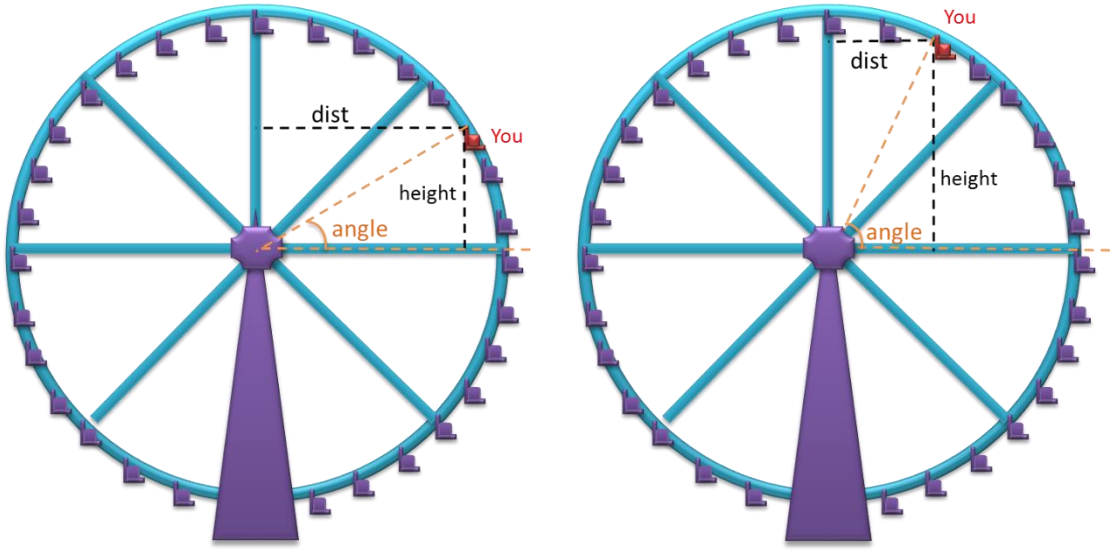
This section concludes the reciprocal family. At this point, we have looked at 3 function families, their inverse families, and their reciprocal families. Now we are ready to study the 4th function family.

4.6 Trigonometric Family

4.6.1 Periodic Functions

Imagine you are on a Ferris wheel like the one shown in the figure below. As the Ferris wheel rotates, what model describes your height at a given time?

A Ferris wheel is different from models we have looked at so far in one very important way. It rotates repeatedly. That is, if you are at the highest (12 O'clock) position now, after some time, you will again reach the same position. This will happen over and over. The time taken to reach the same position is called a "**period**", which is measured in seconds. As a result, functions modeling this sort of periodic activity are called "**periodic functions**". The reciprocal of that period is called the "**frequency**"; that is, how "frequently" you get to the same (e.g., the highest) position (e.g., you reach the highest position twice every minute).



Say we want to model your position on the Ferris wheel at a given time. We can do that if we know the "height" and distance ("dist"), as labeled on the figure above. To find those two values, we will develop two mathematical models.

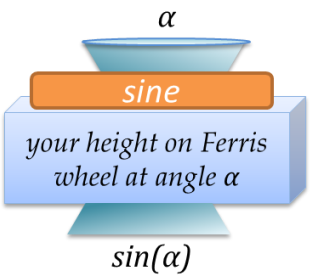
If we want to develop a model to find your height on the Ferris wheel, we can use the angle of rotation as the input to that model. To stick to the convention, we will start measuring this angle when you are at the 3'O clock position. When measuring angles, we consider the counter-clockwise direction to be positive. If we measure in the opposite (clockwise)

direction, that angle created is considered to be negative. Both the starting position and the positive direction are just conventions. We could have chosen the 12'O clock position as the starting point and clockwise as the positive direction without any fundamental changes to our model (as we saw with negative numbers in Section 3.1.2).

With the above conventions, when you are at 12 O'clock, the angle is 90^0 and when you are the 6 O'clock position, the angle is either 270^0, or -90^0, because you could have reached the 6 O'clock position two ways: going 270^0 counter-clockwise, or 90^0 clockwise. As a result, we would expect the same output value (your height) from our model, no matter which of the two angles you use. The same is true for 0^0 and 360^0. We will denote this angle by letter α.

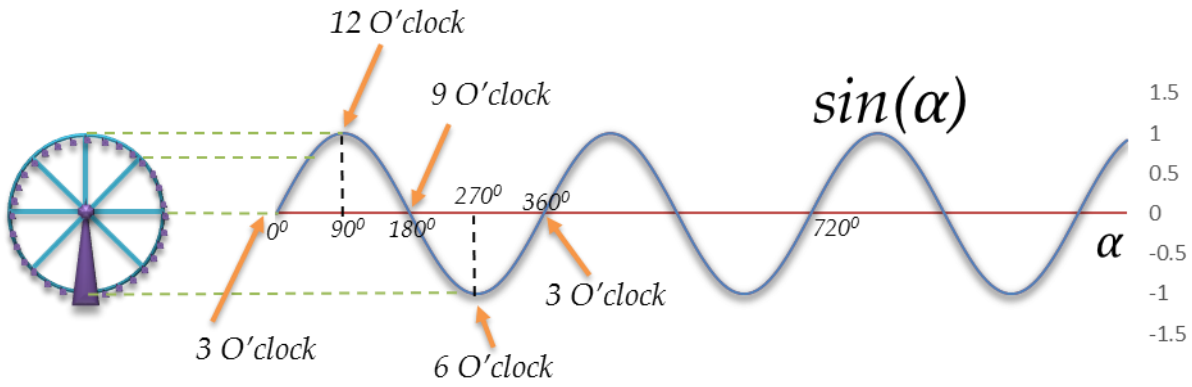
The output of our model is your height on the Ferris wheel. Again, to stick to conventions, we will measure the height from the hub of the wheel, instead from the ground, because the wheel is rotating around the hub. Again, this is just a convention. If we wanted to express the height from the ground, we could simply add a constant offset (height from ground to the hub) to the output. For simplicity, let's say that the radius of the Ferris wheel is just 1 unit (whatever that unit is). So, when you are at 12 O'clock, your height is 1 unit, and when you are at 6 O'clock, your height is -1 unit. When you are at 3 O'clock or 9 O'clock positions, your height is 0 (See figure above).

The relationship between the input angle, α, and your height on the Ferris wheel is given by a model called the **sine** function. We can express this relationship as:



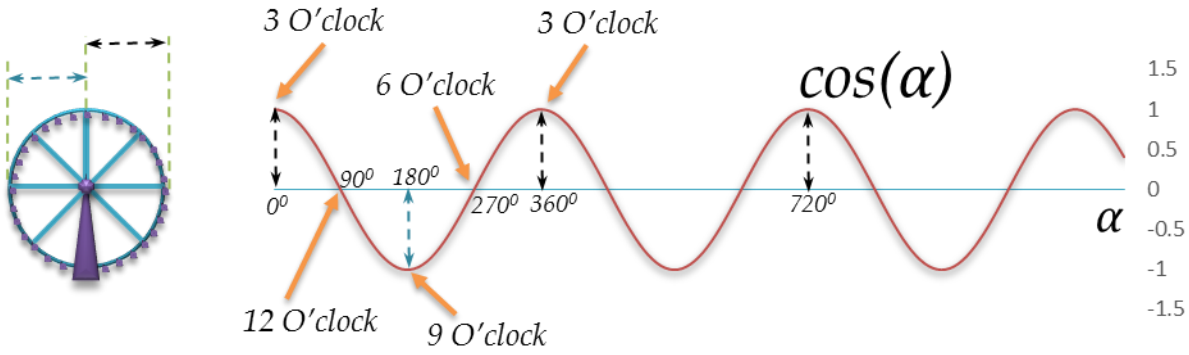
Your height on the Ferris wheel = $\sin(\alpha)$

The following figure shows how your height changes as the Ferris wheel rotates. At the start, when you are at 3 O'clock, angle α is 0 and your height (from the hub) is also zero. As the wheel rotates counter-clockwise, angle α increases and your height starts going up. When you arrive at 12 O'clock (90^0), you have reached the maximum height of 1 unit. When you get to 9 O'clock (180^0), your height reaches zero again. This pattern repeats forever, similar to the way the Ferris wheel keeps on rotating. Study this graph carefully and compare the value of $\sin(\alpha)$ to your height at each of the given angles (positions on the Ferris wheel). It is really important to understand this graph.

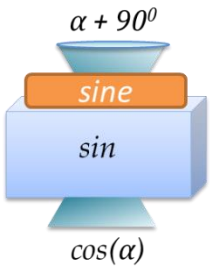


Why is this graph shaped like a wave? It is because the rise of your height is not uniform as you go around. For instance, if you start at 3 O'clock and travel 45^0 counter-clockwise, your height has increased way past the mid-point (in fact, your height is 0.71 units). During the next 45^0, the gain in height is much smaller (0.29 units). This uneven rise in height leads to this wave-like graph.

Now, let's use a second model to model how far you are horizontally from the hub (i.e., your distance, "dist", from the vertical axis of the Ferris wheel). We call this the **cosine** (or cos) function.



The graph of cos(α) has the same shape as the graph of sin(α). The main difference is that at the start, when α is zero, cos(α) is 1, while sin(α) is 0. Essentially, we have shifted the whole sine curve by a quarter of a cycle (90^0) to the left. Because of this, you can find cosine values by looking at the *sine* graph. To find cos(0) all you have to do is look at the output for the input value 90^0 on the sine graph. In general, if we want to find a cosine value at α degrees, we just have to read the value from the sine graph at input $\alpha + 90^0$. We can express this relationship as:



$\cos(\alpha) = \sin(\alpha + 90^0)$

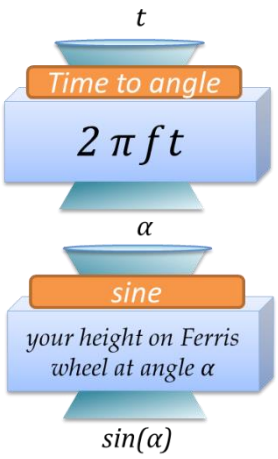
The above means that we can calculate the cosine value for any input angle just by using the sine function. We just have to add an offset of 90^0 to the input angle, as shown by the Visual Model. This offset is called the **phase shift**, because its effect is simply "shifting" the input.

We can define many other functions using the sine and cosine functions. For instance, the ratio between the sine and cosine functions is the tangent (**tan**) function. The reciprocal of sine function is the cosecant function. The main takeaway is this: all of these functions arise from the sine function, which is your height on the Ferris wheel.

As another note, we can measure an angle either in degrees or radians, just like we measure distance in either miles or kilometers. 360^0 is 2π radians, which is equal to the circumference of a unit circle. Therefore, 180^0 is π radians and 90^0 is $\pi/2$ radians. You will see both of these units in practice.

In many real-world applications, it is more useful to have time as the input instead of angle α, because it is easier to measure time. As we know, angle α increases proportionately as time goes by, and hence, α and t are linearly related. For instance, if the Ferris wheel rotates 1 rotation every second, the angle increases 360^0, or 2π radians, every second. After 2 seconds, the angle has increased to $2*360^0$ or $2*2\pi$ radians. Therefore, after t seconds, the angle α is given by

$\alpha = 2\pi\, t$ [angle in radians] or
$\alpha = 360^0 * t$ [angle in degrees].



Instead of rotating 1 rotation every second, if the Ferris wheel is rotating f rotations every second, the frequency is f cycles per second. In this case, angle α increases linearly with f. Why? Because faster the Ferris wheel rotates, the faster the angle changes. Hence, α is given by:

$\alpha = 2\pi f\, t$ [angle in radians] or
$\alpha = 360^0 * f\, t$ [angle in degrees]

Therefore, for a given time, *t*, we can always find the angle α, and sin(α), as shown in the Visual Model. Note that the frequency, f, is treated as a constant. Therefore, if we measure angle α in radians, we can represent the same sine function as a function of time, *t*, as:

$$\sin(\alpha) = \sin(2\pi f\, t) \qquad [\ 2\pi f \text{ is constant }]$$

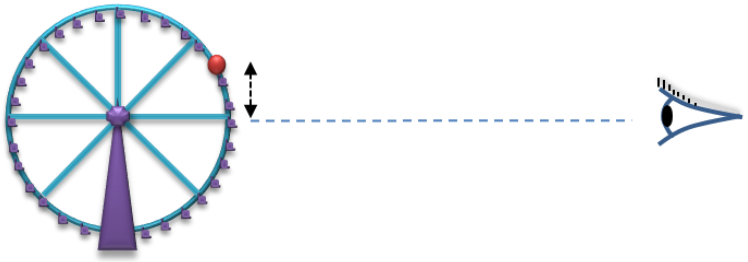
The term 2πf, which is just frequency multiplied by constant 2π, is called the **angular frequency**, often denoted with ω. Angular frequency tells you how much an angle changes per second. For instance, if the Ferris wheel rotates 1 rotation per second, its angle changes 2π radians during that time, and hence, ω is 2π radians per second. Therefore, you can write the above relationship as:

$$\sin(\alpha) = \sin(2\pi f\, t) = \sin(\omega t)$$

You may see all of the above three forms in books. All of them are equivalent ways of expressing the same sine function, which is essentially your height on the Ferris wheel. One form uses an angle as the input. The other two forms use time as the input variable.

You may wonder why we spent so much effort to model your position on a Ferris wheel. What's the purpose of all of this? One word — wheel. During the course of human civilization, the importance of the discovery of the wheel may be second only to the discovery of fire. Walking is linear motion, which is what we are used to. In contrast, wheels rotate. The motion around a wheel, or more precisely around a circle, cannot be analyzed the same way we analyze walking on a straight line. At the same time, this rotation is fundamental to our existence. We exist thanks to the rotation of planets around the Sun (although not in perfect circles). The rotation of wheels in your car takes you places. The rotation of generators produces electricity and the rotation of motors is the main workhorse of industrial civilization, from the tiny motor in your electric toothbrush to the giant motors in factories. The models that help us understand our Ferris wheel, help us understand all these other phenomena.

There is yet another reason for studying rotation, which is not immediately apparent. That is, rotation produces waves. Assume it is dark at night and you are carrying a bright red light on the Ferris wheel. Assume there is someone watching your light from a long distance, in the same plane as the wheel, from the east of the Ferris wheel, as shown below.



As time passes, that observer will see the light move up and down as if you were riding a wave, similar to the sine wave we saw before. In other words, if the observer were to graph your height against time, she would get a sine wave. This is because, one component (vertical component or height) of rotation describes a wave. Similarly, if the observer was above the Ferris wheel looking down, the horizontal component (or distance from the hub) describes a wave. A wave is a natural product of rotation.

The observer, who cannot see the Ferris wheel due to darkness, will observe (and model) the height of your light as a wave (against time). This is exactly how we observe many properties arising at a distance. That is, we don't see rotation but we see the effect of it as a wave. For instance, if you measure the magnitude of voltage (or current) you receive from your electricity company, and plot this amplitude against time, you will get a wave. Why? Because the power company is generating this electricity using rotating generators, like hydro, steam, or wind turbines. You don't see the turbine rotating, but you do see the effect (amplitude of voltage) as a wave. The same happens with electric and magnetic fields. The main difference is that you don't need wires to observe the effect happening at a distance. Your radio station generates an electric field (and a magnetic field), and your radio receiver can detect the strength of it. If your radio station varies this electric field, you can detect that variation from a distance almost immediately. If you graph the variation you observe with time, you will see a wave.

Our lives would be very different without sound waves and electromagnetic waves, which include light. All these phenomena and more can be analyzed using waves. For instance, the music you hear, or any other waveform for that matter, can be expressed as a linear combination of sine and cosine functions, the same way we can express any polynomial as a linear combination of power functions. Now, imagine a bunch of hamsters spinning on differently sized wheels, which are spinning at different speeds (different frequencies). At a given time, if we add up their heights, would you believe that their total height can represent music that comes out of your phone or radio? Well, if you could, now try to get that image out of your mind — little hamsters on wheels creating music in your smartphone.

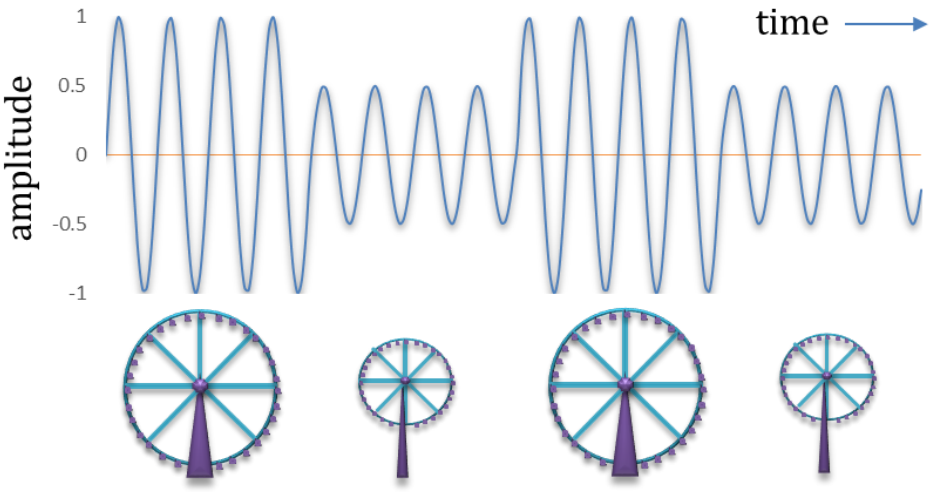
All of these things in life are made possible by rotation, which is why we want tools to model rotation and the resulting waves. If you are still not convinced about the value of trigonometric models and the waves they describe, the next section gives an example of a real-world use that you cannot live without.

4.6.2 Communicating with Waves

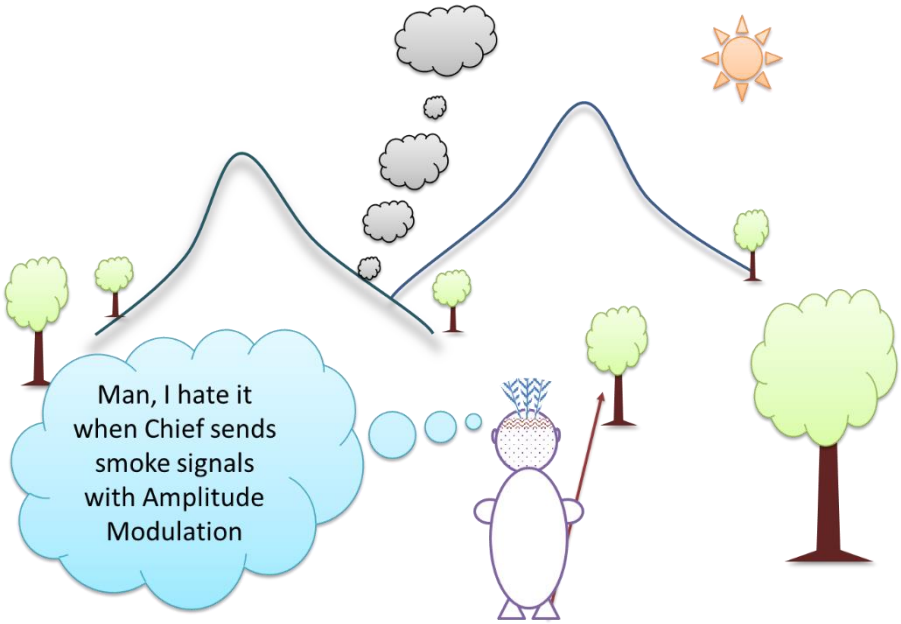
Waves are a primary tool used in communication. You get your radio signals, Wi-Fi, TV, and cellular signals, all thanks to waves. How do we do all that using waves?

Let's start small. Let's take communication using smoke signals. In order to send a message, you need to change some property. For instance, we can use black smoke to convey one message and white smoke to convey another message. In this case, the property we change is the color of smoke. Similarly, if we want to communicate with waves, we need to change some of their properties. What are the properties of waves we can change?

Let's imagine (don't ask me how) that our Ferris wheel can change its radius, every minute. In other words, its radius can grow and shrink. Someone watching from a distance can see the radius of the Ferris wheel changing. Therefore, we can use this change of radius to send a message to the observer. For instance, if we use Morse code, which uses dots and dashes to send messages, a larger radius can represent a dash while a smaller radius can represent a dot. What happens to your height, sin(α), as the radius changes? Your height changes too. This change in height is known as **Amplitude Modulation** or **AM**. The term amplitude refers to the output value of sin(α) function. If your height on the Ferris wheel is large, so is the amplitude. The following figure shows how the amplitude sin(α) changes when we change the radius of the Ferris wheel every minute.



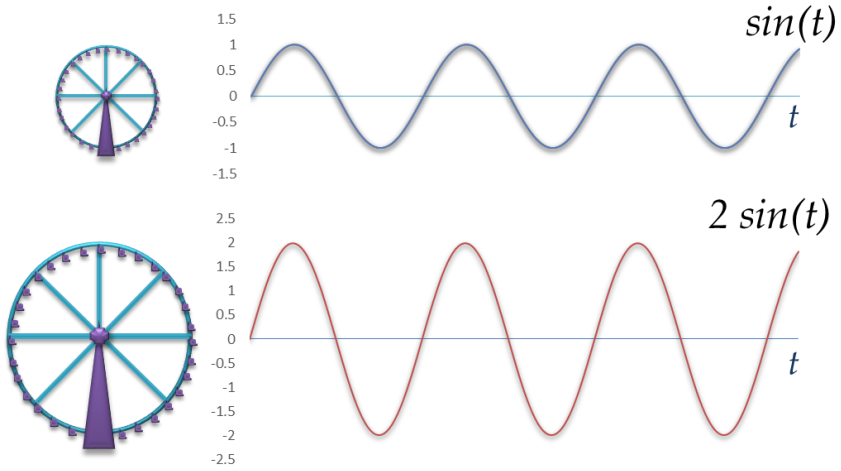
Instead of changing the radius every minute, we can change it every second and send more dots and dashes per minute. If we can change the amplitude continuously, we can communicate a continuously changing value to someone at a distance. This is exactly how you receive AM channels on your radio. The radio station is continuously transmitting a wave of frequency, say 2 MHz (i.e., a sine wave with a frequency of 2 million cycles per second). This is known as the *carrier wave,* and is like a Ferris wheel that rotates continuously without any change to its radius or frequency. The radio station can then change the amplitude of this carrier wave continuously to send any message — for instance, the voice of a weatherman.



The amplitude of a sine wave can be changed by multiplying it by a coefficient (A, in the following example), as:

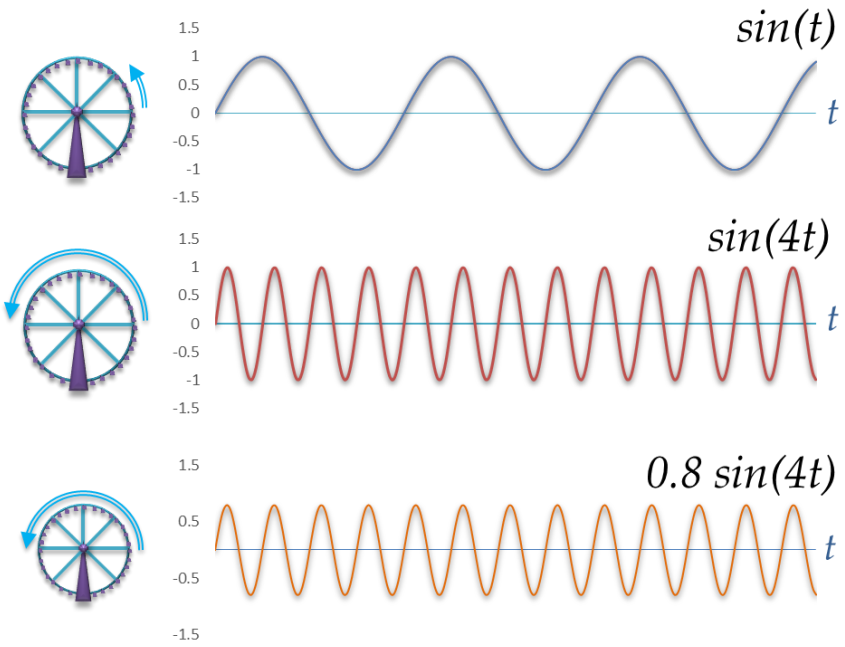
$$s(t) = A \sin(t)$$

Notice that s, which stands for signal, is just the name of the function, which has one input variable t. As an example of different amplitudes, the graphs of $\sin(t)$ and $2\sin(t)$ are given below. Notice the only difference is in the amplitude.



Are there any other properties, besides amplitude, we can change to send a message? Instead of changing the amplitude, what happens if we change the frequency? In our Morse code example, instead of changing the amplitude, to send a dash, let's rotate the Ferris wheel 2 rotations per minute. To send a dot, let's slow down to 1 rotation per minute. This is called **Frequency Modulation,** or **FM**. If we change the frequency every second, we can send more dots and dashes per minute. If we change the frequency continuously by speeding up and slowing down continuously, we can send a continuous message, like the voice of a singer. A radio station can continuously change the frequency of a carrier wave to do that. This is exactly how you receive FM channels on your radio.

How do we model a sine wave with a different frequency? The top graph of the following figure shows $\sin(t)$, and the middle graph shows $\sin(4t)$, which has 4 times the frequency of $\sin(t)$. In other words, $\sin(4t)$ is produced by a Ferris wheel that is rotating 4 times as fast as the Ferris wheel that produces $\sin(t)$. This is shown visually by the blue arrow above each Ferris wheel. During the time it takes the top Ferris wheel to go $1/8^{th}$ of a rotation, the middle Ferris wheel goes ½ a rotation.



The bottom graph of the above figure shows what happens when you change the amplitude and frequency at the same time. The bottom graph, 0.8 sin(4t), models a wave with 4 times the frequency and 80% of the amplitude of the top graph. The general model for such a **waveform** is:

$$s(t) = A \sin(\omega t),$$

where ω is the angular frequency and A is the amplitude. Recall that ω = 2πf, where f is the frequency.

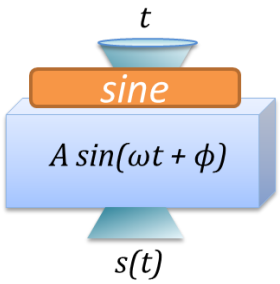
The above model shows how we can change both the amplitude and frequency of a sine wave. There is one additional property we can change in a sine wave: it is the phase shift, which we discussed when we introduced the cosine wave. Remember, we expressed the cosine wave as a sine wave shifted by 90^0 (π/2 radians) as:

$$\cos(\alpha) = \sin(\alpha + \pi/2)$$

where α is measured in radians.

To summarize, a sine wave is characterized by 3 properties:

1) Its frequency, f (or ω, where ω = 2πf)
2) Its amplitude A
3) Its phase shift ϕ



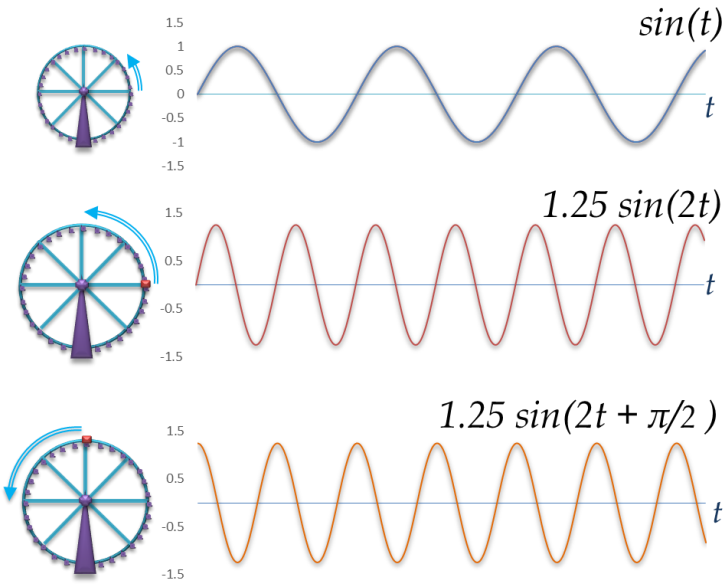
As a result, the general sine function can be modeled as:

$$s(t) = A \sin(\omega t + \phi)$$

or

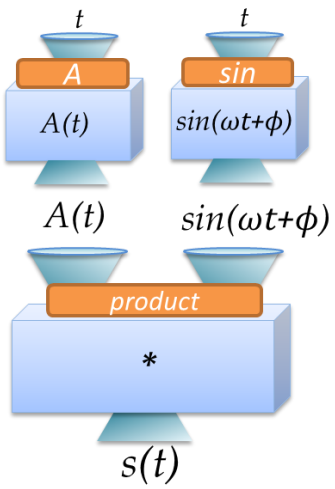
$$s(t) = A \sin(2\pi f\, t + \phi)$$

The following figure shows what this model can represent. On top, we have our original sin(t) function, which can be thought of as your height on our Ferris wheel, with a unit radius and is rotating at angular frequency of 1. The middle curve shows a similar sine wave generated by a Ferris wheel that is rotating 2 times faster with a radius of 1.25 units. The bottom sine wave is generated by a Ferris wheel similar to the middle one, but always leading the middle one by π/2 radians (90⁰) — i.e., one with a phase shift of 90⁰.



The above shows how we can capture any sine wave with the model $A \sin(\omega t + \phi)$. In this model, we treated A, ω, and ϕ as constants. However, they don't always have to be constants. If we vary amplitude, A, with time, we get amplitude modulation. If we vary angular frequency, ω, with time, we get frequency modulation. If we vary phase, ϕ, with time, we get phase modulation (yes, there is such a modulation method). For instance, an amplitude modulated sine wave can be written as:

$$s(t) = A(t)\sin(\omega t + \phi),$$



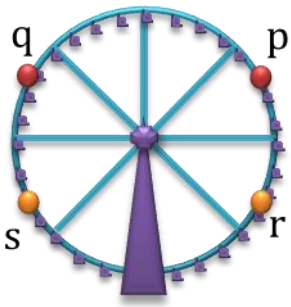
where A is now a function of t. Now you should be able to clearly understand a model like the one above. It says s is a function of input t. Its recipe is a product of two functions, $A(t)$ and $\sin(\omega t + \phi)$, both of which depend on t. Its Visual Model is given on the right. In practice, function $A(t)$ is the "message", or the input signal (e.g., music) that you want to transmit, while $\sin(\omega t + \phi)$ is the carrier wave.

Can you write a similar model for frequency modulation, where ω is a function of t? Can you draw a Visual Model for it? (Hint: Think of function composition)

The above discussion should clearly show you the practical value of the sine function, and its usefulness in modeling real-world relationships. The sine model can represent many different types of waves, where some physical property rises and falls periodically. For instance, when you press a key on a piano, the hammer hits a taut steel string, which causes the string to vibrate, creating different amplitudes (displacement from the resting position of string) at different positions. These amplitudes can be modeled as a wave (a sine wave in particular). Similarly, when a hydroelectric turbine is rotating, the amount of current (and voltage) generated rises and falls repeatedly. This amount (the amplitude) of current varies as a wave, which can be modeled with a sine function. Similarly, if you rotate an electrical charge, the magnitude of force felt due to that electrical charge at some distant point P rises and falls repeatedly and can be modeled as a sine wave. Similarly, if you ride a Ferris wheel, your height rises and falls as a wave, allowing us to model it using a sine wave. This is why we were so obsessed with your height on the Ferris wheel, because, it is the same model we use to model many real-world properties that change periodically.

4.6.3 Inverse and Reciprocal Models of Trigonometric Functions

Since the function $f(x) = \sin(x)$ outputs a value between -1 and 1 for any given input angle, the inverse of the sine function, called **arcsine**, outputs an angle for any input between -1 and 1. Using our Ferris wheel analogy, if we use your height on the Ferris wheel as the input, the arcsine function would output the angle you make with the horizontal axis. The **arccos** function is defined similarly, as the inverse of the cosine function.



Since trigonometric functions are periodic functions, the input domain has to be restricted for an inverse function to exist. For instance, your height on the Ferris wheel is the same when you are at points p or q in the figure shown. Similarly, whether you are at r or s, your height is the same. Therefore, we cannot have both p and q (or r and s) in the domain of the sine function, if we want an inverse function. As a result, we have to restrict the domain of the sine function to points in the first quadrant (where p is) and the 4th quadrant (where r is), which are angles between -π/2 to +π/2. Using a similar argument, for the cosine function, we have to restrict the domain from 0 to π, because points p and r (and q and s) have the same horizontal distance from the hub.

The reciprocal function of $\sin(x)$ is $1/\sin(x)$, which is known as **cosecant**(x). Similarly, the reciprocal function of $\cos(x)$ is $1/\cos(x)$, which is known as **secant**(x).

We are finally done with the trigonometric family, the 4th function family we studied. For each of those families, we looked at different family members, their inverses, and their reciprocals. Before we look at how we can combine all 4 families to build more complex models, we will briefly look at one other heavily used function — the factorial function.

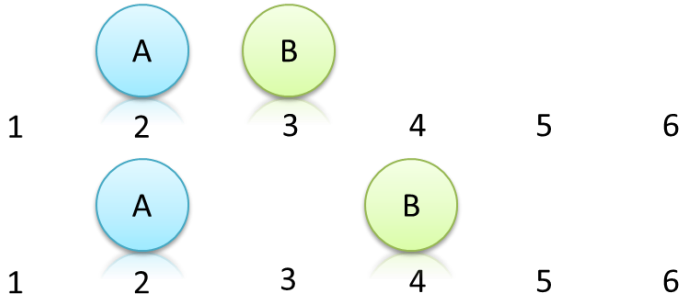
4.7 Growing Faster than Exponential: The Factorial Function

If you thought exponential functions grew really fast, wait until you meet the **factorial** function. With exponential functions, the output of the previous step got multiplied by a constant base (a constant multiplier). In the factorial function, the output of the previous step gets multiplied by a variable input. As the input grows, the multiplier grows as well:

$$\text{factorial}\ (n) = 1 * 2 * 3 * \ldots * n$$

For instance, factorial(3) is 1∗2∗3, which is 6. Similarly, factorial(5) is 1∗2∗3∗4∗5, which is 120. Similarly, factorial(10) is 3,628,800. The change in output when the input changed from 5 to 10 is incredible! By definition, factorial(0) is 1.

Factorial function is useful in calculating permutations and combinations. For instance, if there are 6 balls of different colors, how many different ways can they be arranged on a table with 6 positions? Well, we have 6 positions to be filled with 6 balls. Ball A can go into any of the 6 positions. Thus, there are 6 ways to place ball A. For each of those positions, ball B can be placed in 5 positions. The following figure shows two such arrangements, when ball A is at position 2. Ball B is shown at position 3 in the first arrangement, and at position 4 for in the second arrangement. By extending this argument, you can see that, when Ball A is at position 2, Ball B can go into any of the 5 positions 1, 3, 4, 5, 6.



Since ball A can go into any of the 6 positions, and for each of those positions ball B can go into any of the remaining 5 positions, there are 6∗5 ways to place ball A and B. Similarly, after placing ball A and B, ball C can go into any of the remaining 4 positions. Thus, balls A, B, and C can be placed in 6∗5∗4 ways. Similarly, ball D can be placed at any of the remaining 3 positions, and ball E can be placed at any of the remaining 2 positions. After placing ball E, there is only one position left, and ball F has to go there. As a result, altogether, there are 6∗5∗4∗3∗2∗1 ways to arrange the 6 balls. As you can see, this is factorial(6).

Since the factorial function is useful in calculating permutations and combinations, it is quite useful in evaluating probabilities. We will see more uses of the factorial function in power series expansions in Chapter 5.

Computer code for the factorial function is given below. Notice that we are using recursion, the same strategy we used to calculate pushups with exponential growth. In the code below, we are evaluating `factorial(10)`.

```
function factorial( n )
{
      if (n == 0)
            result = 1
      else
            result = n * factorial( n - 1 )

      return result
}
```

```
y = factorial( 10 )
print( y )
```

Within the code, factorial(10) gets evaluated using the statement:

```
result = n * factorial( n - 1 )
```

Note that "`result`" is the output of `factorial(n)`. If n=10, to evaluate factorial(10), we multiply 10 by factorial(9) as:

$$\text{factorial}(10) = 10 * \text{factorial}(9) \qquad \textbf{(A)}$$

How do we find factorial(9)? It is evaluated as:

$$\text{factorial}(9) = 9 * \text{factorial}(8) \qquad \textbf{(B)}$$

If we put (A) and (B) together, we get:

$$\text{factorial}(10) = 10 * 9 * \text{factorial}(8)$$

Similarly, factorial(8) is evaluated as 8 ∗ factorial(7). This process continues until we reach factorial(0), which is defined as 1. Therefore, factorial(10) evaluates to:

$$\text{factorial}(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 * 1$$

The factorial function naturally leads to recursion for the same reason exponential model naturally led to recursion: in both models, to find the output for step n, we use the result from the step $n-1$. In fact, to find the result for step n, we multiply n by the result of step $n-1$. That's what's expressed with the statement:

```
result = n * factorial( n - 1 )                    (1)
```

Note that "`result`" is the output of the current step (for step n). Using result of step $n-1$ to calculate the result for step n leads to rapid growth. In the exponential model, at each step, we multiply the result of step $n-1$ by the same number (e.g., by 2, or by e) as:

```
result = 2 * exponential( n - 1 )                  (2)
```

Compare (2) with (1). In factorial function, we multiply the result of step $n-1$ by an ever-increasing n (i.e., by 5 for step 5, by 6 for step 6, etc.) That's why the factorial function grows much faster than the exponential function.

Now, you should be able to compare the growth of the power function, x^n, the exponential function, e^x, and the factorial function, $n!$. Out of these three, factorial function grows the fastest, followed by the exponential function, followed by the power function.

Since the factorial function increases really fast, the reciprocal of the factorial function decreases really fast. This 1/factorial function is a key ingredient of important relationships, as we will see in Chapter 5.

4.8 Building Larger Models from Simpler Models

So far in this chapter, we have met 4 function families, their inverse functions and their reciprocal functions. These functions serve as the basic building blocks of real-world models. There are several common ways you can build more complicated models using these basic functions:

1) Composition of two functions
2) Multiplication of two functions
3) Division of two functions (Multiplication by reciprocal)
4) Linear combination of functions

You can use any of the above techniques, or any combination of these techniques, to *build* a complex model. Notice that function division similar to $f(x)/g(x)$ is equivalent to *multiplying* $f(x)$ by the *reciprocal* of $g(x)$. This is one of the reasons why we learned about the reciprocal functions.

Similarly, an existing complicated model can be *analyzed* using the same techniques. As an example, let's look at a common function we see in statistics, called the **Normal Distribution**, or **Gaussian Distribution**.

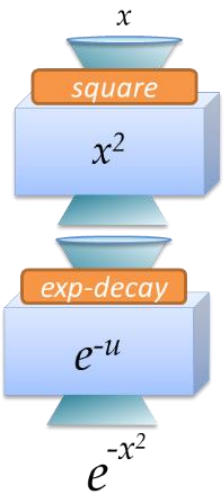
There is a pattern to how some properties, like the height of people in a country, are distributed. For instance, when you walk down the street, you are more likely to meet a person of average height (or around that) than an extremely tall or short person. Therefore, we call height a normally distributed property (or informally, height has a "bell-shaped" distribution).

The simplest form of a "**bell curve**" takes the following form, where x is a property like height of a person.

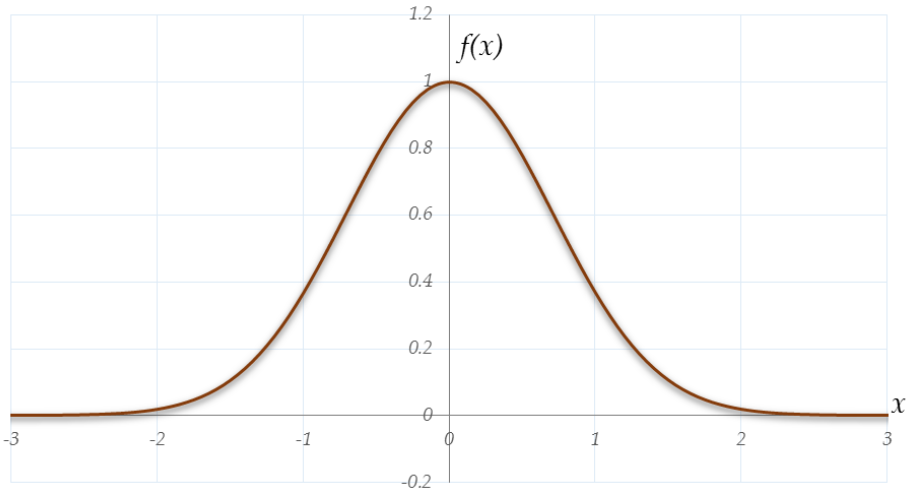
$$f(x) = ke^{-x^2} \qquad \textbf{(1)}$$

where k is a constant. Can you identify this model? If we replace x^2 by another variable, u, this becomes:

$f(u) = k\, e^{-u}$ where $u=x^2$

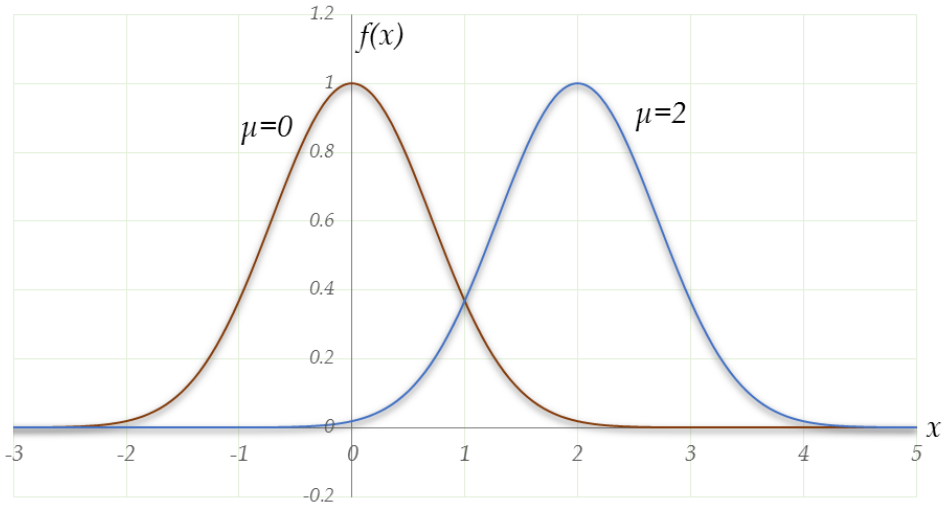


You know what this model is. In terms of input u, this is an exponential decay model. In terms of x, it says that the square of the input (x^2) decays exponentially. Notice that this is *function composition,* as shown in our Visual Model. Since x^2 is always positive, the above function f has the same output value whether we feed in x or $-x$ as its input. Therefore, f is symmetric about the y-axis. The graph of f is shown below for k=1.



Notice that the peak value of 1 occurs when u=0, because e^0 is 1. When u=0, x=0 as well. However, this is not the general normal distribution function you meet in the statistics class: it uses $u = (x - \mu)^2$ instead of $u=x$, where μ is a constant and represents the mean of the distribution. At which input does the peak occur in this distribution? We saw that the peak occurs when u=0. What's the value of x when u=0? For u to be zero, x must be equal to μ. This means, that the peak of this general model occurs at x=μ. In other words, instead of a bell curve centered around 0, we now get a curve centered around μ (i.e., symmetric around μ), as shown below. The following figure shows the normal distribution curves for both μ=0 (gray) and μ=2 (blue). As expected the blue curve is shifted right by 2 units.

One more thing: the general normal distribution given in (1) has a k term. This is simply a scaling factor, where every output value is multiplied by k.



The complete model for the normal (or Gaussian) distribution is given by the following, seemingly frightening formula. However, as we saw, it is quite easy to understand if we use the right tools.

$$f(x) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Note that μ and σ are constants. Therefore, as we saw above, we can write this as:

$$f(u) = ke^{-u} \quad \text{where } u = k_2 (x - \mu)^2$$

where k and k_2 are constants. With suitable function composition, you can immediately identify that the above is based on the exponential decay model, in terms of input u (or in terms input x^2).

This example shows how you should approach complicated algebraic models. Seemingly daunting models can be usually decomposed into much simpler models.

The other important lesson is that you can use function composition, function multiplication, and function division (which is multiplication by reciprocal) to build up new models.

Let's look at a few other examples. Can you describe each of the following models?

(a) $\sin^2(x) + 2\sin(x) + 1$
(b) $1 / (1 + e^{-x})$
(c) $\sin(x)\cos(x)$
(d) $x \sin(x)$

(e) e^{2x} - $3e^{x}$ + 2

(f) tan(x)

Let's start with (a). Can you recognize this as function composition? Hint: use variable u to represent sin(x). Then, (a) becomes $u^2 + 2u + 1$, which is a polynomial. This can be factorized to $(u+1)^2$. If we substitute $v = u + 1$, this becomes v^2, which is a power function. If we had to find the roots of this function, the roots would be v=0, which means that u=-1, which in turn means that sin(x) = -1. In short, if we feed sin(x) into the polynomial function $(u + 1)^2$, we get (a). Alternatively, if we add 1 to sin(x) and feed that result into the quadratic power function v^2, we get (a).

You should be able to immediately identify (b) as a reciprocal function. First, we start with exponential decay function, e^{-x}, and then add 1, which shifts the function up by 1 unit. Then, we take the reciprocal of all of that. When we take the reciprocal, high output values become low and low output values become high. Function (b) is called the *logistic growth model,* and it models how a population (e.g., a mosquito population in a region), grows under constraints.

You should see (c) and (d) as the product of two functions, in the form of $f(x) = g(x) * h(x)$. If we have to find the roots of this product to solve the equation $f(x) = 0$, you can treat $g(x)$ and $h(x)$ as factors. Therefore, the roots of $g(x)$ and roots of $h(x)$ become the roots of the combined function $f(x)$.

Do you see the similarity between (e) and (a)? Again, this is function composition. If you substitute $u = e^x$ in (e), you would get the polynomial $u^2 - 3u + 2$, which can be factorized as $(u-1)(u-2)$. If we had to find the roots, the roots are u=1 and u=2, or $e^x = 1$ and $e^x = 2$, which occur at x=0 and x=ln(2).

We already saw that tan(x) is function division in the form $f(x) = g(x)/h(x)$, where $g(x)$ = sin(x) and $h(x)$ = cos(x). If we had to find the roots of $f(x)$, we have to look at the roots of $g(x)$, which are roots of sin(x). Similarly, if we have to find the values at which $f(x)$ is not finite, we have to look at roots (zeros) of $h(x)$, or cos(x).

Isn't it fascinating how you can build more complex models from the simple functions we learned about? Consequently, when we are given a more complicated model, we should always try to think about how that model can be composed from simpler functions.

4.9 Summary of Function Families

The following table summarizes the 4 function families we met in this chapter:

1) Power function family: the input n is raised to the n^{th} *power*. The inverse is the n^{th} *root* and the reciprocal is $1/x^n$ (or x^{-n}).
2) Polynomial function family: a sum of power functions. The reciprocal is just 1/function. For the linear function, the inverse is a linear model as well. The general polynomial model may not always have inverse functions.
3) Exponential function family: a given base is exponentiated by the input. The inverse is a *logarithmic* function, whereas the reciprocal is an *exponential decay* function.
4) Trigonometric function family: represents the linear projections (height/distance) of a rotating object. The basic members of this family are sine and cosine. Their reciprocals are cosecant and secant, respectively. The inverse functions of this family outputs angles.

Function Family	Important Member Function	Inverse of Member Function	Reciprocal of Member Function
Power			
	x	x	$1/x = x^{-1}$
	x^2	$x^{1/2} = \sqrt{x}$	$1/x^2 = x^{-2}$
	x^n	$x^{1/n} = \sqrt[n]{x}$	$1/x^n = x^{-n}$
Polynomial			
	$ax + b$	$(x - b)/a$	$1/(ax + b)$
	$ax^2 + bx + c$	$-b \pm (b^2-4a(c-x))^{1/2} / 2a$ †	$1/(ax^2 + bx + c)$
Exponential			
	2^x	$\log_2(x)$	2^{-x}
	10^x	$\log_{10}(x)$	10^{-x}
	e^x	$\ln(x)$	e^{-x}
Trigonometric			
	$\sin(x)$	$\arcsin(x)$	$1/\sin(x) = \text{cosec}(x)$
	$\cos(x)$	$\arccos(x)$	$1/\cos(x) = \sec(x)$

† Note that this is an inverse relation and not one inverse function, since it describes two inverse functions as we saw in Section 4.4.2.2.

In addition to these broad families, we met the factorial function, where positive integers from 1 to n are multiplied together to produce the output for an input n. We also looked at linear combinations, which are just linear functions of more than one input.

4.10 The Story So Far

In the previous chapters, we saw how functions came into existence, their common properties, their common problems, and the objects they consume and produce. This chapter was dedicated to looking at different families of the function dynasty, and what they are good at.

If you have ever seen a construction crew building a house, you would have noticed a wide array of talent. There are carpenters, electricians, roofers, plumbers, masons, architects, structural engineers, supervisors, painters, and so on. Similarly, building models that can represent the real-world requires a diverse set of talents. In this chapter, we saw how functions step up to the plate to fulfill that demand.

We explored the function families in a methodical way. Primarily, we looked at 4 families: power functions, polynomial functions, exponential functions, and trigonometric functions. For each of those families, we looked at its inverse and reciprocal families. This gave us a cohesive view on how all functions are related.

As you saw, some functions were pretty simple fellas. For instance, a power function just makes its input "more powerful" by multiplying its input by itself a given number of times. However, when many such simple power functions are added together (linearly combined), the result is a polynomial, which is a much more capable model.

Then, we encountered exponential functions, which can make input expand over a large range, and their inverse functions, logarithmic functions, which can compress an input into a small range. Similarly, we saw the reciprocal model of exponential functions, which model exponential decay.

As our 4^{th} family, we looked at a class of functions that are useful in representing rotation or periodic behavior. We saw how these functions can be used to model waves, and how they can model communication based on waves.

Just when we thought power functions grew fast, we met exponential functions. When we thought those were growing super-fast, we met the factorial function, which grew even faster.

We also saw how all of these different types of functions model real-world phenomena we study in physics and engineering. In particular, we looked at a technique for analyzing functions with multiple inputs as linear functions with just one variable input and output.

Finally, we looked at how we can combine simpler functions to produce much more complex functions, that can model the real-world relationships that we are interested in.

5 SERIES: FUNCTIONS UNLIMITED

Chapter Overview: This chapter introduces you to the concept of a series, where we combine multiple terms, sometimes an infinite amount, to come with a series of terms. We will see how such series can approximate functions we have seen before, uniting seemingly unrelated function families.

In Section 4.2.4, we saw how any polynomial function can be represented as a series of power functions. A polynomial function has a finite (limited) set of terms. For instance, take the following cubic function:

$$f(x) = 2x^3 + 9x^2 - 2x + 1 \qquad \textbf{(1)}$$

How was that cubic function made? Well, we built up a function using a series of 4 terms (i.e., as a sum of 4 different power functions). In other words, a series is another way to build a more complex function using simpler functions. For a polynomial, this does not come as a real surprise. But can we do the same for other functions?

Before we can answer this question, we need to get accustomed to series notation. Let's start with the familiar polynomial model.

5.1 A Polynomial as a Series

An n^{th} degree polynomial has at most n+1 terms. In other words, an n^{th} degree polynomial is a series (a sum) of n+1 terms, as shown below.

$$f(x) = a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + \ldots + a_Nx^N \qquad \textbf{(2)}$$

Take a look at each term. For instance, let's look at a_2x^2, which is a power function. This term represents $9x^2$ in (1) above. In a_2x^2, a_2 is the coefficient and x^2 is the input raised to the second power (i.e., the input is squared). A polynomial is a series (a sum) of such terms, where the power of the input can be from 0 to n — that is, from x^0 (which is the constant term) to x^n. Make sure you understand this before we move on.

The above series can be written using Sigma function (summation operation). We use the Greek letter Sigma to represent this summation operation, as follows:

$$f(x) = \sum_{n=0}^{N} a_n\, x^n$$

It is very important to understand the above notation. Let's start with $a_n x^n$. Let's say that n is 2. Then, this term becomes $a_2 x^2$, as we saw in (2). It means, a_2 is the coefficient and x^2 is the power of the input (in this case, the input is squared). If a_2 is 9, this term represents $9x^2$. The Sigma notation says that n starts at zero and goes until N (notice the notation above and below the Sigma sign). That is, n starts at 0 and increments one by one, until it reaches N. When n goes from 0 to N, we get all N+1 terms. That is, when n=0, we get the constant term, $a_0 x^0$, when n=1, we get the linear term, $a_1 x^1$, and when n=2, we get the quadratic term, $a_2 x^2$. We continue until n is equal to N. When n is equal to N, we get the N^{th} term, $a_N x^N$. We sum up all of these terms (Sigma means sum), to get the function $f(x)$ on the left-hand side.

If N is equal to 3, $f(x)$ is a 3rd degree polynomial with 4 terms. If N is equal to 2, $f(x)$ is a quadratic polynomial with only 3 terms. It is extremely important to understand Sigma function (operator) before we move on.

A polynomial is a series of power functions

The computer code for evaluating a series is given below. To start, let's assume that all coefficients are 1. If we evaluate `sigma(2, 4)`, the code will evaluate the polynomial:

$$f(x) = x^0 + x^1 + x^2 + x^3 + x^4$$

at x=2. This will evaluate to 1+2+4+8+16 = 31.

```
function sigma( x, N )
{
      sum = 0
      for n=0 to N
      {
            term = power( x, n )
            sum = sum + term
      }
      return sum
}
```

```
y = sigma( 2, 4 )
print( y )
```

In our `sigma` function, we start by initializing the variable "`sum`" to zero. The variable `sum` is like an empty bag, and we will add each `term` to this bag as we go on. Next, we have a new structure called a '**for loop**'. This for-loop represents Sigma notation nicely. Remember, how Sigma notation instructs `n` to take integer values from 0 to N? That's exactly what the

loop statement '`for n=0 to N`' does. It takes the variable `n` from 0 to N, one step at a time. At each step, we evaluate one term of the polynomial as we do with the Sigma notation. For instance, when `n`=2, we calculate the x^2 term.

To calculate the power of a given term, we get help from another built-in function, named "`power`", which we also used for exponentiation in Section 3.4. For instance, when `n` is 3, we call power (2, 3), to calculate 2^3. Once we calculate the value of a term, we add it to the `sum` (our bag). That's what Sigma operator (summation) does as well. The statement "`sum = sum + term`" tells just that. It just says to put another `term` into our bag holding the current `sum`. For instance, at the start, "`sum`" is zero. In the very first step, the value of "`term`" is 1, because `power(0,1)` is 1. So, we have `sum` = 0 + 1, and hence the new value of "`sum`" becomes 1. We continue this loop until `n` becomes equal to `N` (i.e., until we calculate all n+1 terms and add them to our bag).

It is quite illuminating to see the similarities between Sigma notation and the "for loop" in computer code. You can see the same overall structure in both forms. Mainly, we add a sequence of terms, from n=0 to N, to produce a sum. The computer code should help you look at Sigma notation as a function (or operator) with a recipe to generate N terms and add them together.

Sigma operator is modeled by a "for loop"

If you are a computer programmer, please note that we don't follow the syntax of a particular computer language precisely but you should be able to change it to the syntax of your favorite language quite easily. What's really important is understanding the connection between math and computer code.

In the computer code we just looked at, we used a built-in `power` function, which evaluates x^n. However, it is quite easy to write your own power function for non-negative integer powers by multiplying x by itself n times, as shown below. This also uses a "for loop". We start with `product` equal to 1. In each step (iteration) of the loop, we multiply the current value of `product` by `x`. In the first step, we multiply `product` by `x`, which results in `x`. In the next step, we multiply this result by `x` again, resulting in $x*x$. We continue this for n steps, to get x^n.

```
function power( x, n )
{
      product = 1                    // initialize product to 1
      for n=1 to N                   // n goes from 1 to N
      {
            product = product * x    // multiply product by x
      }
      return product
}
```

The `sigma` function we wrote above assumes that all coefficients are 1. That's rarely the case. To fix this issue, we can store the coefficients in an **array**. Think of an array as a set of slots, each holding one value. If the name of the array is '`coefficients`', the expression `coefficients[0]` represents the 0th slot of the array, `coefficients[1]` represents the 1st slot of the array, and so on. The array shown below stores 4 coefficients to represent the coefficients of the polynomial $5 + 3x + 4x^2 + 2x^3$.

5	3	4	2
coefficients[0]	coefficients[1]	coefficients[2]	coefficients[3]

```
function sigma( x, N, coefficients )
{
      sum = 0
      for n=0 to N                   // n goes from 0 to N
      {
            term = coefficients[n] * power( x, n )
            sum = sum + term         // each term is added to sum
      }
      return sum
}
```

```
coefficients = [5, 3, 4, 2]          // 5 + 3x + 4x² + 2x³
y = sigma (2, 3, coefficients)       // evaluate at x=2
print( y )                           // prints 43
```

The new code that accepts a coefficient array is shown above. Now, the function uses the corresponding coefficient, `coefficients[n]`, to calculate the n^{th} term. That's all.

Notice how we call (use) the above sigma function. First, to start with, we put values 5, 3, 4, 2 into the coefficients array to represent coefficients of $5 + 3x + 4x^2 + 2x^3$. Then, we call the sigma function to evaluate this cubic function at x=2. The print statement should print 43, since $5 + 3x + 4x^2 + 2x^3$ at x=2 is 5+6+16+16 = 43.

5.2 The Sky is the Limit: Power Series

In the previous section, we represented a polynomial function with a series of a finite number of terms. This raises the question whether we can represent other types of functions, in addition to polynomials, using a series. As we will see shortly, the answer is yes. However, there is one catch. Most of the time, we will need an **infinite** number of terms, compared to the finite number of terms we need for a polynomial.

If I ask you to name an infinite series that you use every day, you might say, "Wait! I don't use any infinite series — I hardly know any". That's not true. If you buy a Latte, for $3.99, or buy gas for $3.499, you are using a system built on infinite series: decimal numbers.

To start, let's see how we can express a whole number (an integer) using the decimal notation. Let's take number 5897. It can be represented as:

$$5897 = 7*1 + 9*10 + 8*100 + 5*1000$$

In general, we can express a decimal number as:

$$a_0*1 + a_1*10 + a_2*100 + a_3*1000 + \ldots$$

You can immediately notice that 1, 10, 100, 1000, etc. are powers of 10. Therefore, we can write 5897 as:

$$5897 = 7*(10)^0 + 9*(10)^1 + 8*(10)^2 + 5*(10)^3$$

Therefore, in general, we can write an integer in decimal notation as:

$$a_0(10)^0 + a_1(10)^1 + a_2(10)^2 + a_3(10)^3 + \ldots + a_n(10)^n + \ldots \quad \textbf{(1)}$$

Notice that if we replace 10 with x, (1) would be a polynomial of x with an infinite number of terms. Therefore, we can write an integer as a function of x as follows:

$$f(x) = a_0 x^0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots \quad \textbf{(2)}$$

When x=10, we get the decimal notation.

The above is an **infinite series**, because in a decimal number, we can have an infinite number of digits. The term $a_n x^n$ represents the n^{th} digit. Recall that x^0 is 1, because any number raised to the power of zero is 1.

Note that the right-hand side of (2) can be represented more succinctly using Sigma notation:

$$f(x) = \sum_{n=0}^{\infty} a_n \, x^n$$

The right-hand side represents an infinite series, because there are an infinite number of terms on the right-hand side. We call this series a **power series**, because each term is a different power of input, x, and there is an infinite number of terms.

A decimal number is a Power Series

In computer code, here is how you can represent 5897 with the sigma function we wrote above:

```
coefficients = [7, 9, 8, 5]          // coefficients of 5897
y = sigma( 10, 4, coefficients )     // 4 terms with x=10
print( y )                           // prints 5897
```

Quick quiz: if we wanted to write a representation of any binary number using Sigma notation, how would you do that? You guessed right. We just have to use x=2 instead of x=10. Therefore, (2) represents the formula for writing a whole number in any base.

Another quiz: how would you write a decimal fraction, like 0.6879, using (2)? One easy way is to use x=0.1. Note that powers of 0.1 gives 0.1, 0.01, 0.001, 0.0001, etc. Similarly, you can write any binary fraction with x=½. As an example of fractional representation, here is the computer code to produce fraction 0.3547891.

```
coefficients = [3, 5, 4, 7, 8, 9, 1]
y = sigma( 0.1, 7, coefficients )          // x=0.1, 7 terms
print( y )                                 // prints 0.3547891
```

What if we wanted to represent a decimal number that has both a whole part and a fractional part, like 53.234? We can extend (2) to do that. Can you think of a way? Hint: notice that 10^{-1} = 0.1 and 10^{-2} = 0.01. Now you should be able to see it. Yes, we should allow n to be negative as well. Therefore, we can represent any decimal number with the following function f, when x=10.

$$f(x) = \sum_{n=-\infty}^{\infty} a_n x^n$$

Time for a quiz: what's an example of a decimal number with an infinite number of digits? A decimal number like 32.3214 does not need an infinite number of digits. However, irrational numbers like π or $\sqrt{2}$ do.

As you can see, an infinite power series is not such an exotic concept as it looks at first glance.

5.3 Maclaurin and Taylor Series

In the previous section, we saw how we can use an infinite series to model a function that produces decimal numbers. Can we express other functions in the same way, and if so, is there a way to find the terms of the series that represent those functions?

The answer to both of those questions is yes. Let's look at the easiest method to do this first: **Maclaurin series**. The Maclaurin series models a function as a series of power terms, similar to the model describing the decimal notation. Remember, polynomials are a series of power terms, but the number of terms is finite. You can look at the Maclaurin series as an "*infinitely long polynomial*". Therefore, the Maclaurin series of a given function $f(x)$ can be written as:

$$f(x) = a_0 x^0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots \qquad \textbf{(3)}$$

$$f(x) = \sum_{n=0}^{\infty} a_n x^n$$

In the Sigma notation given above, the only difference from a polynomial is the upper bound. Now it is infinite. The terms remain the same. This is exactly what we saw in the previous section when representing an integer decimal number. So, what's different?

If a Maclaurin series is an "infinitely long polynomial", in order to get a function, all we have to do is find the coefficients, a_n, of this infinite series. Different coefficients represent different functions. Maclaurin series tell us exactly how to find those coefficients. Let's look at an example.

Let's say we want to find the power series expansion of e^x. If such a power series existed, using (3), we could write:

$$e^x = a_0 x^0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n + \ldots \quad \textbf{(4)}$$

If we can find each coefficient a_n, then we get the power series expansion of e^x. Now, how do we find each coefficient? Finding a_0 is easy. If we evaluate equation (4) at x=0, we get:

$$e^0 = a_0 0^0 + a_1 0^1 + a_2 0^2 + a_3 0^3 + \ldots + a_n 0^n + \ldots \quad \textbf{(5)}$$

We know that e^0 and 0^0 are 1. All other powers like 0^1, 0^2, …, 0^n are just zero. Thus, we get:

$$1 = a_0$$

Yay! We found the coefficient of the 0^{th} term, a_0. That was quite easy. Because a_0 was the coefficient of the power function x^0, evaluating (4) at x=0 was all that was required to find a_0. However, we cannot simply continue this process and find a_1, because a_1 appears as the coefficient of power function x^1, as $a_1 x^1$. However, if there were a way to make that power function x^0, giving us the term $a_1 x^0$, then, we could set x=0 to find a_1. But, how on earth can we get power function x^0 from power function x^1? That's where calculus comes in. To do that, we have to find the *slope* of a function. If you know calculus, that's the **derivative** of the function. If you haven't taken calculus, just look at it as the *slope* of the function at a given point, as you would find from its graph.

Since (4) is an equation, we have to apply the same operation to both sides. Therefore, we find the slope of both sides of (4). Let's start with the left side. What's the slope of e^x? From Section 4.3.1, we know that the slope of the natural exponential function is the same as the output of e^x. That is, the slope (derivative) of e^x is e^x itself!

Now, we have to find the slope of the right-hand side of (4), which is a sum of power functions. From calculus, the slope of the power function x^n is given by another power function, nx^{n-1}. Notice that x^{n-1} has a power that is one lower than x^n. For instance, the slope of x^2 is given by function $2x^1$. Similarly, the slope of power function x^1 is $1 \cdot x^0$, or just 1, which we can easily see from its graph — i.e., the function $f(x) = x$ is a line with a slope of 1. Furthermore, the slope of x^0 is 0, because x^0 is just 1 — i.e., $f(x) = 1$ is a horizontal line, which has a slope of 0. Therefore, by using the slope (derivative) of both sides of (4), we get:

$$e^x = 0 + 1 a_1 x^0 + 2 a_2 x^1 + 3 a_3 x^2 + \ldots + n a_n x^{n-1} + \ldots \quad \textbf{(6)}$$

Now, if we evaluate the above slope at x=0, we get:

$$e^0 = 0 + 1 a_1 0^0 + 2 a_2 0^1 + 3 a_3 0^2 + \ldots + n a_n x^{n-1} + \ldots \quad \textbf{(7)}$$

Again, we know that e^0 and 0^0 are 1, and all other powers like 0^1, 0^2, …, 0^n are just zero. Thus, we get:

$1 = 1 \cdot a_1 \qquad \Rightarrow \qquad a_1 = 1 / 1$

Yay! We found the coefficient of the 1st term too. Now, to find other coefficients, we can repeat this entire process of taking the slope of both sides and setting x=0. If we take the slope (derivative) of (6) and set x=0, we get:

$1 = 1 \cdot 2 \cdot a_2 \qquad \Rightarrow \qquad a_2 = 1 / (1 \cdot 2)$

By repeating this process again and again, we get:

$1 = 1 \cdot 2 \cdot 3 \cdot a_3 \qquad \Rightarrow \qquad a_3 = 1 / (1 \cdot 2 \cdot 3)$

$1 = 1 \cdot 2 \cdot 3 \cdot a_4 \qquad \Rightarrow \qquad a_4 = 1 / (1 \cdot 2 \cdot 3 \cdot 4)$

In general, for the n^{th} term, we get the coefficient:

$1 = n! \, a_n \qquad \Rightarrow \qquad a_n = 1 / n!$

Putting all the coefficients together, the power series expansion we get for e^x is given below. Notice that, in essence, what we have is an infinite polynomial (an infinite number of power functions). The coefficient of the n^{th} term is given by the reciprocal of the factorial of n.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^n}{n!} + \cdots$$

In Sigma notation, we can write this as:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Now, you may ask - why will we ever need to know any of this? Well, have you ever thought about how your calculator or computer calculates $e^{3.015}$? This is exactly how they do it. Of course, calculators and computers don't evaluate an infinite number of terms. They stop after a given number of terms and get a very reasonable approximation with a very small error. The beauty of this method is that you can always decrease the error by calculating more terms.

Why do you think we can ignore the higher order terms? There is a clue in the coefficients. Which model represents the denominator of the coefficient? It's the factorial function. Remember how quickly the factorial function grows? Remember how it grows faster than the power function and the exponential function? So, what happens to each term when the

denominator gets super large due to the factorial? The coefficient approaches zero really fast.

As an example, let's write computer code for the Maclaurin series expansion of e^x and use that to approximate the value of $e^{2.01}$, with 5 terms and 10 terms. Recall that we wrote computer code for the factorial function in Section 4.7.

```
function exp( x, N )
{
      sum = 0
      for n=0 to N-1
      {
            term = power( x, n ) / factorial( n )
            sum = sum + term
      }
      return sum
}
```

```
y = exp( 2.01, 5 )              // e^2.01 using 5 terms
print( y )                      // prints 7.0636
y = exp( 2.01, 10 )             // e^2.01 using 10 terms
print( y )                      // prints 7.4630
```

The above function `exp` calculates e^x using N terms of the Maclaurin expansion. As expected, each term is calculated as a fraction between x^n and factorial(n). As n increases, the factorial(n) becomes much larger than x^n, which makes '`term`' smaller and smaller. If you calculate $e^{2.01}$ using a calculator, you can see that the value we get with 10 terms is quite accurate. However, if we use only 5 terms, we are close but not accurate enough for most purposes. The Maclaurin series, thankfully, allows us to get higher precision by doing more work.

Similar Maclaurin series expansions are available for $\sin(x)$, $\cos(x)$, and many other functions. Since we need calculus to derive them, we will just list the expansions here. The Maclaurin series for $\sin(x)$ is very similar to the series of e^x, with only two differences: (i) only odd powered terms are present, and (ii) the coefficient of every odd term is negative. Thus, we get:

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Note that the coefficient of every odd (1st, 3rd, 5th, etc.) term is negative (we start counting terms from zero, when n=0, so x is the 0th term). The series for cos(x) is similar to the series of e^x, with only two differences: only even powered terms are present, and every odd term has a negative coefficient, just as in the series for sin(x). Note that the 0th term, 1, can be thought of as $x^0/0!$, because x^0 is 1 and 0! is defined to be 1.

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

This can be written compactly using Sigma notation as:

$$\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$$

As you can see, we express the alternating negative sign using $(-1)^n$, because even powers of -1 are positive and odd powers of -1 are negative. Additionally, the term $2n$ appears as the exponent and input to the factorial function, because only even terms are present. Quick quiz: how would you modify the above sigma notation to express sin(x)? Remember, with sin(x), only odd terms are present. Thus, instead of using $2n$ above, we have to use $2n$+1.

As you may have noticed, we obtained the series for sin(x) and cos(x) from the series for e^x. This relationship will become really useful in Section 6.6. Functions like e^x, sin(x), and cos(x) are called **transcendental** functions because we need an *infinite* series of terms to express them. In contrast, functions that can be expressed in a finite number of basic algebraic operators (addition, multiplication, square root, …) are called **algebraic** functions. Therefore, transcendental functions are non-algebraic.

The computer code for evaluating the value of the cosine function using N terms of the Maclaurin series is given below. For this evaluation, it may help you to look back at the Maclaurin series for cos(x) in Sigma notation. Again, we calculate the sign of each term using $(-1)^n$. A variable called `two_n` is used to represent $2n$. The variable `two_n` is used as an input to the power function (for x^{2n}) and the factorial function (for $(2n)!$).

```
function cos( x, N )
{
      sum = 0
      for n=0 to N-1
      {
            two_n = 2 * n
            sign = power( -1, n )
            term = sign * power( x, two_n ) / factorial( two_n )
            sum = sum + term
      }
      return sum
}
```

```
y = cos( 1.33, 10 )            // cos(1.33) using 10 terms
print( y )
```

The more general form of the Maclaurin series is called the Taylor series. Hence, some texts may refer to the above expansions of e^x, $\sin(x)$, and $\cos(x)$ as Taylor series expansions. To find the coefficients of series, instead of finding the slopes at input 0 (i.e., x=0), Taylor series finds the slopes of functions at some arbitrary input value a (i.e., x=a). This helps find the series expansions of functions that are not defined at 0.

Now you should be able to fully apricate the value of power series and in particular the value of Maclaurin and Taylor series. In the real-world, they give us an easy way to evaluate transcendental functions like e^x, $\sin(x)$, $\cos(x)$, using calculators and computers. At a more theoretical level, these series hint at us a connection between the exponential function and trigonometric functions. We will explore this in the next chapter.

5.4 Fourier Series

The Maclaurin and Taylor series can be used to build up functions using power functions (terms). Quite importantly, they show how more complicated functions can be built up using simpler functions. Conversely, they also show that more complicated functions like e^x or $\sin(x)$ can be broken down (decomposed) into an infinite number of simpler functions (power functions).

The **Fourier series** does the same thing for periodic functions (e.g., any waveform). Instead of using polynomial terms to build up functions, Fourier series uses sine and cosine functions to build up any waveform. That is, any periodic waveform can be built up using an infinite number of sine and cosine functions with different frequencies and amplitudes.

We will not go into mathematical details of Fourier series. However, Fourier series is yet another example demonstrating the power of infinite series and how we can use infinite series to model the real world.

5.5 The Story So Far

In the previous chapters, we learned about the function dynasty: how functions came into existence, their common properties, their common problems, the objects they consume and produce, and their most popular families. In this chapter, we looked at how an infinite sum of simpler functions can model much more complex functions.

You may have seen a brawny guy moving a heavy object. However, a group of kids may be able to do the same. That's the power in numbers. If you have an infinite number of increasingly more powerful guys, that group can take the place of one really powerful person. We saw that with infinite series, where an infinite sum of simpler functions can represent a much more sophisticated function. In particular, we learned about the power series, which is essentially an infinite sum of power functions, which can represent transcendental functions like exponential and trigonometric functions.

Even something as mundane as a decimal number is the result of a power series. You may know that ancient Romans could not figure out this decimal representation of numbers, because they couldn't figure out the power series (positional value). That's why Roman numerals and their number system is not as useful as the decimal or binary representations.

The ability to approximate a more powerful function using a sum of simpler functions has ginormous practical implications. That's exactly what your calculator does whenever you press e^x. That's what the world's most powerful supercomputers do when they have to find e^x or $\sin(x)$ or any other transcendental function for complicated calculations. You will see another application of infinite series in the next chapter, when we examine Euler's Identity.

Just as you can construct a more complex function by combining multiple power functions together, by adding many sine functions together, you can construct any waveform. This is the basis of the Fourier series and Fourier transforms. All of the digital music you hear, digital movies you watch, and cell phone calls you receive, are made possible by these fundamental mathematical models.

SECTION II: BEYOND FUNDAMENTALS

6 FUNCTIONS THAT CAUSE (YOUR HEAD TO) SPIN

Chapter Overview: This chapter expands the concept of a number to represent two-dimensional objects. We will see how that allows us to build richer models, especially that support rotation, with the help of our old friend, multiplication operator.

We use numbers to abstract real world objects. When we say we have 10 cows, we are using the number 10 to abstract the cows regardless of whether each cow is sitting, sleeping, munching, heading east, heading west, or jumping over the moon. This abstraction works really well for this case (counting cows). But does it always work?

Let's start with a puzzler. There are 10 identical cannons. A cannonball shot from each cannon falls exactly 1 mile from the cannon (not even an inch off). A shooting range has an X marked on the ground. Each cannon is brought on to this X mark (one after the other) and ordered to fire 3 cannonballs. If we keep a target exactly 1 mile from point X, how many cannonballs will hit the target?

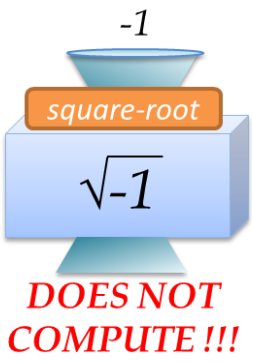
Since there are 10 cannons and each fires 3 cannonballs, 30 cannonballs must hit the target. Right? Wrong! I omitted one crucial detail: although each cannon has to fire from X, the cannon can be pointed in any direction (e.g., east, west, northeast, north of northeast, or in any other direction). Therefore, it would be lucky coincidence if any of the cannonballs hit the target.

If you think the above puzzler is pure obfuscation, think a little bit harder. The problem is deeper than that. The number 10, which we used to represent 10 cannons, did not do a good job at representing 10 cannons, because a cannon can be rotated and pointed in any direction. We figured this out the hard way — when none of our cannonballs hit the target.

If number 10 cannot represent 10 cannons faithfully, which number can do that? We ran into a similar situation, mathematically, in Section 3.1.2. Remember, when we walked 3 steps forward and 5 steps backward? We found out that whole numbers just failed to represent the outcome of that exercise. So, we had to expand our definition of numbers to include negative numbers. A similar situation with division forced us to expand our notion of numbers to include fractions.

But the situation with cannons is even harder than that. We can use negative numbers to represent the "opposite" direction, but not *any* direction. We will soon encounter a number that can do just that. Our failed attempt at representing the real-world with real numbers will again reveal the answer.

As you know, when we square some number x, we get x^2. We call x the square-root of x^2. The square-root function is the inverse of the square function. The problem is that x^2 is always positive. Hence, if we want to find the square-root of a negative number, we can't find one, because there is no number that gives a negative number when squared. Simply put, there is no number that can represent the square root of a negative number, such as $\sqrt{-1}$.



When faced with such a hurdle, we can do the same thing we did before when we encountered negative numbers and fractions: we invent a new number and see where that takes us.

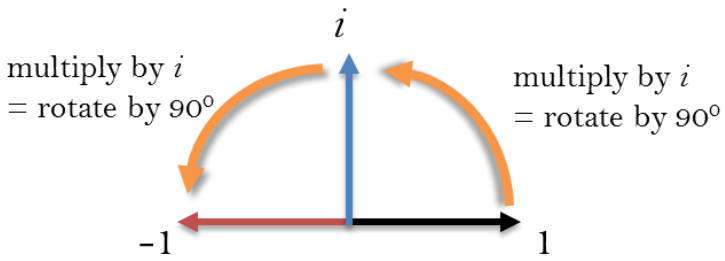
To investigate this "new number", let's start by giving it a name. Let's call this new number i, which is equal to $\sqrt{-1}$. What do we know so far about this number? We only know its definition — i is a number that when squared, will give us -1. That's all we know. Let's write down what we know:

$$i * i = i^2 = -1$$

What does this mean? Well, i^2 behaves like -1. But how does -1 behave? As a multiplier, -1 causes reflection (rotation by 180^0), as we discussed thoroughly in Section 3.3.2. We can write this as follows:

i^2 ⇨ rotation by 180^0
$i * i$ ⇨ (some effect) $*$ (some effect) **(1)**

According to (1), if multiplying by i causes "some effect", when that "effect" is repeated, we should get rotation by 180^0. With a little bit of thinking, we see that this "some effect" is rotation by 90^0. This is because if we rotate something by 90^0, and again rotated that result by 90^0, we get rotation by 180^0, as shown below.



To verify that multiplying by i rotates a number by 90^0, let's look at several examples. Let's start with what we know. We know that:

$i^2 = -1$

What about i^3?

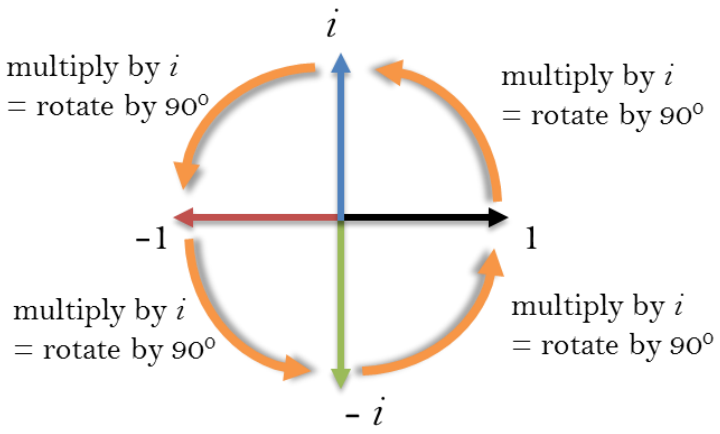
$i^3 = i^2 * i$

$= -1 * i = -i$

What about i^4?

$i^4 = i^3 * i = -i * i = -(i^2) = 1$

We can summarize multiplication by i using the following figure.



As evident from the above figure, multiplying by i causes a number to rotate by 90^0, counter-clockwise.

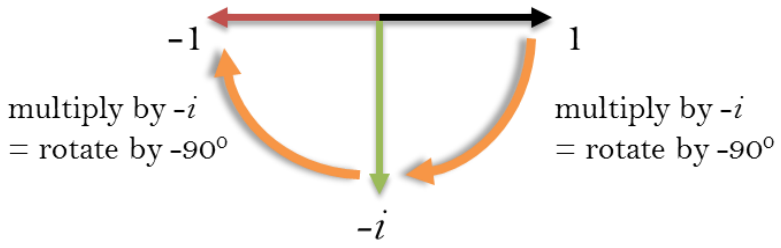
Multiplying by i causes rotation by 90^0

Similarly, what does multiplying by $-i$ do?

$1 * -i = -i$

$-i * -i = i^2 = -1$

Therefore, multiplying by $-i$ causes rotation by 90^0, *clockwise*. As we discussed before with trigonometric functions, we represent clockwise angles as negative angles. Therefore, this is a rotation by -90^0.



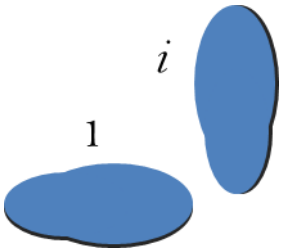
Multiplying by -i causes rotation by -90°

Does it look like we have a number that can represent a cannon in our puzzler? Not quite, but we are getting there.

6.1 A "Rotated" Number

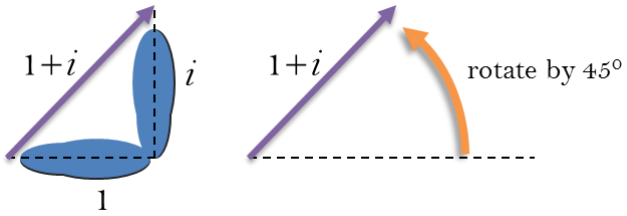
When we encountered negative numbers for the first time, we wanted to see how it would behave with our fundamental operators: addition, subtraction, multiplication, and division. If i is like any other number, we should be able to do operations like addition, subtraction, multiplication, and division with it.

Let's start with addition. If we add 1 to i, we will have to represent the result as $1+i$. We cannot simplify it further than that because the two numbers do not have the same orientation (rotation). However, $i+i$ can be added together to yield $2i$, because they have the same orientation.



Here is another way to look at a number like $1+i$. How do we describe 5+7? We go 5 steps (say forward), and then go another 5 steps, in the same direction (forward). How about 5 + (-3)? We go 5 steps forward and 3 steps backwards. Then, what about $1+i$? We go 1 step forward, then turn 90° (turn left), and go 1 step forward in that direction. That is, go one step forward and one step to the "left". Where we end up can be represented by $1+i$, as shown in the figure.

The number $1+i$ has one real part (the real number 1) and one part that is rotated by 90° counterclockwise, relative to the real part. We call this rotated part the "**imaginary**" part. Note that the word "imaginary" is a misnomer — there is



nothing imaginary about it. If going one step forward is real, so is going one step to the left. Therefore, it is better to refer to the imaginary part as the "90^0 rotated" or the "orthogonal" part.

As shown in the above figure, $1+i$ as a whole is rotated by 45^0, because we went one step forward and one step to the left. From geometry you learned in school, $1+i$ makes a 45^0 angle with the real (horizontal) part, as shown above.

To get a little bit more intuition about this number, let's look at $1+i$ in a different way. Let's see what happens if we square $1+i$.

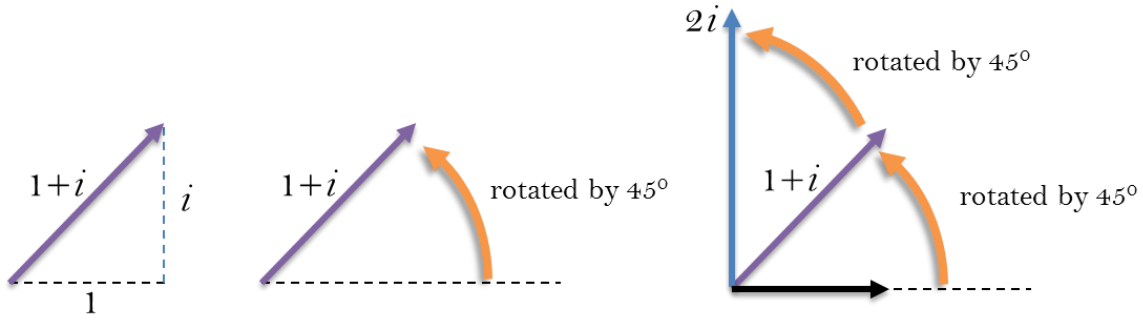
$$(1+i)^2 = 1 + 2i + i^2 \quad = 2i \quad \text{(because } i^2 \text{ is -1)}$$

To derive the above, we used the distributive property as follows:

$$(1+i)\,(1+i) \quad = 1(1+i) + i(1+i) \quad = 1 + i + i + i^2 = 2i$$

What does this mean? $1+i$ multiplied by itself yields the imaginary (or rotated) number $2i$. Which means $1+i$ is the *square root* of $2i$. Therefore, we can deduce the following:

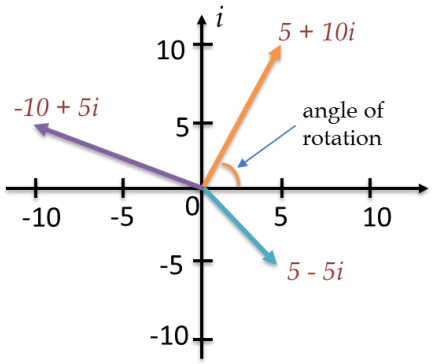
1. $1+i$ must have a rotation of 45^0 with respect to the real part. This is because the square of $1+i$ produces $2i$, which we know makes a 90^0 angle with the real part
2. The magnitude, or the length, of $1+i$ is $\sqrt{2}$, because when squared $1+i$ produces $2i$, which has a magnitude of 2.
3. The square root of an imaginary ("rotated") number, seems to be an imaginary number itself.



Because of the 1st observation above, we can draw $1+i$ as shown on the left figure above, making a 45^0 angle with the real part. Because of the 2nd observation, we know that $1+i$ has a magnitude of $\sqrt{2}$. Incidentally, we get $\sqrt{2}$ if we apply the Pythagorean formula to the left most figure, because both i and 1, have a magnitude of 1. The right figure shows how $1+i$ acts as the square root of $2i$.

A number like 1+*i*, which has both a real *and* an imaginary part, is called a **complex number**. A general complex number can have any number of real and imaginary parts. For instance, 3 + 2*i* has 3 real parts (3 steps forward) and 2 imaginary parts (2 steps to the left). Therefore, we can represent a general complex number as *a* + b*i*, where *a* is the real part (*a* steps forward) and *b* is the imaginary part (*b* steps to the 'left').

In general, we can use orthogonal axes to represent real and imaginary parts of a number, because 1 and *i* are orthogonal (90^0 rotated). These two orthogonal axes define a two-dimensional (2D) plane, which is often referred to as the **complex plane**. Just as a real number can be placed on a number *line*, any complex number can be drawn on this complex *plane*. Several complex numbers on the complex plane are shown in the figure.



The above discussion showed us some examples of complex numbers. To see the true nature of a complex number, as we did with other numbers, we have to apply basic arithmetic operators to them. That's what we will do next.

6.2 Functions with Complex ("Rotated") Input and Output

6.2.1 Basic Arithmetic Operators (Functions)

As with any other number, we can perform addition, subtraction, multiplication, and division with complex numbers. When we are doing addition, we have to add the real parts and imaginary parts separately. Why? Remember, a complex number like 5 + 3*i* represents two orthogonal directions: forwards and backwards (the real part), and "left" and "right" (the imaginary part). Since we're working with different directions, we have to add the two parts separately. Subtraction follows the same logic. For instance,

(5 + 3*i*) + (2 + 9*i*) = 7 + 12*i* [addition of complex numbers]

(5 + 3*i*) – (2 + 9*i*) = 3 – 6*i* [subtraction of complex numbers]

The following computer code shows how to add two complex numbers. To represent the two parts of a complex number, we use an array of two elements, with the 0^{th} element representing the real part and the 1^{st} element representing the imaginary part. In function `complexAdd`, first we create a new two-element array to represent the result. Then we add real and imaginary parts separately, to produce the result. That's it. As an aside, if you are a programmer, you will notice that it is more elegant to create a structure or a class to represent a complex number than a two-element array.

```
function complexAdd( x, y )          // x=a+bi   y=c+di
{
      result = new Array[2]          // create new complex number
      result[0] = x[0] + y[0]        // add real parts (a+c)
      result[1] = x[1] + y[1]        // add imaginary parts (b+d)
      return result
}
```

```
num1 = [5, 3]                        // 5 + 3i
num2 = [7,-1]                        // 7 - i
result = complexAdd( num1, num2 )    // add up two numbers
```

When we multiply two complex numbers, we use the distributive property, as we saw in multiplying $(1+i)$ by $(1+i)$. Thus,

$$(5 + 2i) * (3 - i) = 5(3 - i) + 2i(3 - i) \quad \text{[multiplication of complex numbers]}$$

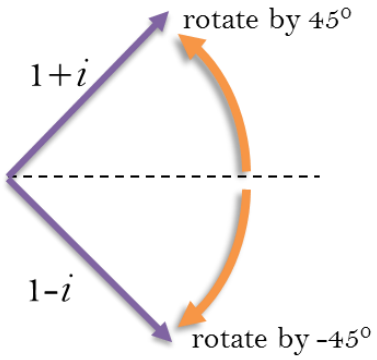
$$= 15 - 5i + 6i - 2i^2$$

$$= 17 + i$$

How do we divide two complex numbers? Since we don't know how to divide by a number with 2 parts, we use a special trick to make the denominator only one part (i.e., real). For instance, if we have $(5 + 3i)/(1+i)$, we can multiply both the numerator and denominator by $(1-i)$. Since $(1+i)(1-i) = 2$, we can readily divide any complex number by the real number, 2. In the above example, $1-i$ is called the **conjugate** of $1+i$. In order to find the conjugate of a complex number, you just flip the sign of the imaginary part.

Can you explain why multiplying a complex number by its conjugate always leads to a real number? Algebraically, $a + bi$ and $a - bi$ are conjugates, because:

$$(a + bi) * (a - bi) = a^2 + b^2$$



The result is a real number — no imaginary part. There is an intuitive way to explain this as well. The angle that $1+i$ and $1-i$ make with the real axis is the same except $1-i$ makes a negative angle (-45^0) while $1+i$ makes a positive angle (45^0). Therefore, if we multiply $1+i$ by $1-i$, you can view this operation as rotating $1+i$ by -45^0 (or rotating $1-i$ by $+45^0$). The result is a number with a rotation of 0^0, which is a real number. In general, if we

have a complex number $a + bi$, its conjugate is $a - bi$ because $a - bi$ makes the same angle as $a + bi$ but in the opposite direction.

Conjugates $a + bi$ and $a - bi$ have the same magnitude but opposite rotation

Computer code for multiplying two complex numbers is shown below. Again, we calculate the real part of the output and imaginary part of the output separately. Can you modify this function to do complex division? (Hint: write two formulas for calculating the real and imaginary parts and notice that both get divided by the same real number.)

```
function complexMult( x, y )                // x=a+bi  y=c+di
{
      result = new Array[2]
      result[0] = x[0]*y[0] - x[1]*y[1]   // ac - bd  (real)
      result[1] = x[0]*y[1] + x[1]*y[0]   // ad + bc  (imaginary)
      return result
}
```

```
num1 = [5, 3]                          // 5 + 3i
num2 = [7,-1]                          // 7 - i
result = complexMult( num1, num2 )     // multiply two complex nums
```

6.2.2 Complex Multiplication as Two Operations

In Section 3.3.2, we saw multiplication of real numbers as a combination of two operations:

(1) scaling (or amplification), and
(2) reflection (rotation by 180^0)

As we saw in this section, multiplication by a complex number, generalizes operation (2) further. Essentially, complex multiplication performs two fundamental operations:

(1) scaling (or amplification), and
(2) rotation by any angle

This should help you see complex numbers as a generalization of real numbers. To summarize, if one of the inputs to the multiplication operator (function) is

- a complex number (z), then the other input is scaled by the magnitude of z and rotated by z's angle

- a negative number (x), then the other input is scaled by the magnitude of x and reflected (rotated by 180^0)
- a positive number (x), then the other input is scaled by the magnitude of x (no rotation at all)

Complex multiplication is a generalization of scalar multiplication

Squaring, and more generally exponentiation, is similar to multiplication. Exponentiation with an integer exponent can be thought of as repeated multiplication (that is, repeated rotation and repeated scaling). For instance, if we have,

$(1+i)^5$

This does two things. First, exponentiation raises the magnitude of $1+i$, which is $\sqrt{2}$, to the power of 5 — i.e., $(2)^{5/2}$. Second, exponentiation rotates the number 5 times. Remember that $1+i$ has a rotation of 45^0. Repeated rotation done 5 times produces $45^0 * 5 = 225^0$ of rotation relative to the positive real axis.

What about fractional exponents? Let's start with square-root. It is the inverse of squaring. For instance, if some complex number z is the square root of $1+i$, then $z^2 = 1+i$. This means that z should have half the rotation of $1+i$ and the square-root of the magnitude of $1+i$. Other roots work in the same way.

Now, it's time to revisit our cannon puzzler. We can use a complex ("rotated") number to represent any of the cannons in our puzzler. Not only that but we can use these "rotated" numbers to rotate the cannon by any angle in 2D, just by multiplying it with the appropriate complex number.

Not only have we found a way to represent rotated objects in 2D, but we also found a way to rotate objects in 2D as we please.

We can represent rotated objects! We found a way to rotate objects!

6.3 Complex Roots of Polynomials

One place where complex numbers arise naturally is in the roots of polynomials. Recall that roots are solutions to the equation $f(x) = 0$. Using roots, we can represent a polynomial as a product of its factors, which are linear functions. In Section 4.2.4.2, we saw that the root of each linear function is a root of the polynomial itself.

We also learned about a theorem that gives us the number of roots in a polynomial: the *Fundamental Theorem of Algebra*. It states that an n^{th} degree polynomial has n roots (including repeated roots), where each root can be real or complex.

What does this mean? How can we get complex numbers as roots from a polynomial with real coefficients? Let's look at a real-world example to understand what's going on.

Say you have a cube (like a Rubik's cube). Say its volume is 8 units (this unit could be any cubic unit, like cubic-inches or cubic centimeters). We want to find the length of a side of this cube. How would we do that? If the length of side is x, we can model this relationship as, $x^3 = 8$. In standard polynomial form, we can write this as:

$$f(x) = x^3 - 8 = 0$$

You can immediately see that 2 is a root of $f(x)$, because $f(2)$ is zero. Obviously, that means if a side is 2 units, the volume is 8 units. Are we done then? Not quite. The Fundamental Theorem of Algebra says that there are two other roots. Since x=2 is a root, $(x - 2)$ is a factor. If you remember polynomial division, by dividing $x^3 - 8$ by $(x - 2)$, we get $x^2 + 2x + 4$. Thus, we can write:

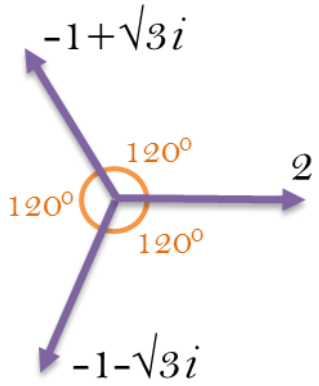
$$f(x) = (x - 2)(x^2 + 2x + 4)$$

What are the roots of $x^2 + 2x + 4$? Using the quadratic formula, we get:

$$x = -1 + \sqrt{3}i \quad \text{and} \quad x = -1 - \sqrt{3}i$$

As you can see, both of these roots are complex. But, what does it mean to have complex roots to our real-world problem? How can the side of a cube be $-1 + \sqrt{3}i$?

To start, let's draw $-1 + \sqrt{3}i$. One thing you can immediately see is that its magnitude is 2. Using the Pythagorean theorem, the length squared is $(-1)^2 + (\sqrt{3})^2$, which gives us a magnitude of 2. Now, that's a good start. It has the same length (magnitude) as the real root. So, what's the difference between the real root and $-1 + \sqrt{3}i$? You guessed right: as our picture shows, it is rotated; it is rotated by 120^0.



We also know that, if we cubed this root, the result should be 8. Let's do that. First, we square this root by multiplying it by itself. Thus, we do $(-1 + \sqrt{3}i)(-1 + \sqrt{3}i)$. Remember what happens when we multiply: the output gets scaled (magnified) and rotated. When we multiply a number by $-1 + \sqrt{3}i$, it gets magnified by 2 and rotated by 120^0. Since we multiply $-1 + \sqrt{3}i$ by itself, we get another complex number that has a magnitude 4 and a

rotation of 240^0. This complex number is $-2-2\sqrt{3}i$. If we multiply this by $-1+\sqrt{3}i$ to find the cube, we get 8. This is summarized below:

$-1+\sqrt{3}i$ [magnitude 2 and rotation 120^0]

$(-1+\sqrt{3}i)^2 = -2-2\sqrt{3}i$ [magnitude 4 and rotation 240^0]

$(-1+\sqrt{3}i)^3 = 8$ [magnitude 8 and rotation 360^0 (or 0^0)]

What in the world does this result mean? It means that whether we start with a side of length 2 or a side of $-1+\sqrt{3}$i, we end up with a cube of volume 8 units. The same reasoning applies for the other root, $-1-\sqrt{3}$i, which is rotated -120^0.

The original root said that the length of a side is 2. However, that was not the whole story. What if we had a side of length 2 that was rotated by 120^0 to begin with? That side also leads to a cube with a volume of 8 units. If we considered only the real root, we are ignoring this fact. The complex roots reminded us of this fact. A cube with a side of 2 units and another cube with a side of $-1+\sqrt{3}i$ units are two different objects, if we were to consider their rotations. Both of these cubes have the same volume, but if we consider their rotation, they are two different objects. Therefore, all three roots represented three different cubes with a volume of 8 units.

This example should make the real-world meaning of complex numbers crystal clear. Still you may object that you don't really care about the rotation of a cube when you are only concerned about its volume. That's fair. However, many problems will require you to take rotation into account. Remember what happened when we ignored the rotation of cannons in our puzzler?

6.3.1 Why do They Come in Pairs?

In the above example for $f(x) = x^3 - 8$, the complex roots came as a pair: $-1+\sqrt{3}i$ and $-1-\sqrt{3}i$. As you can see, they are a conjugate pair. That is, one is rotated by 120^0 and the other is rotated by -120^0. This is not unique to this example. Complex roots always come in a conjugate pair. If you think a little bit, you should be able to explain this from what we know about complex numbers. Hint: what happens when you multiply a complex number by its conjugate? The result is a real number, because the rotations cancel out.

A polynomial with real coefficients always has a real output when the input is real. That's why you can always graph a polynomial with two real axes. However, we know that a polynomial is always equivalent to the product of its factors (linear functions). The factors have the same roots as the polynomial as we discussed in Section 4.2.4.2. So, assume that one of those factors has a complex root. What does this factor do? We know that multiplying by a complex factor results in rotation. Thus, if we multiplied by one complex

factor, the output is no longer real (it is rotated, so the output is complex, or rotated). For the output of the entire polynomial to be real, there has to be another factor that would "undo" the rotation done by the first factor. What would undo the rotation done by multiplying by a complex number? Its conjugate!

As you can see, our intuition of complex numbers helped us understand a deeper fact about roots of polynomials. If you understood that, here is an easier problem. Can you prove that a polynomial of odd degree must have at least one real root? For instance, $f(x) = x^3 - 8$ had real root 2. Why would we always have a real root for a polynomial of an odd degree? That's easy: it's because complex roots come in pairs. For instance, a 5th degree polynomial can have a maximum of 4 complex roots (2 pairs). The remaining root cannot be complex, because complex roots have to come in pairs. Thus, there is at least one real root.

As we saw in Section 4.2.4.2, the roots of a polynomial completely characterize that polynomial. That's why roots are so important, in addition to roots being the solution to the polynomial equation $f(x) = 0$. So, to recap, what does it mean for a function to have complex roots? It means that the solutions to the equation $f(x) = 0$ can be rotated objects. Further, if an objected rotated by α degrees clockwise is a solution, another object rotated counter-clockwise by the same α degrees is also a solution. In other words, solutions to any real-world relationship modelled as a polynomial do not prefer clockwise rotation to counter-clockwise rotation.

6.4 Graph of a Complex Function

Graphs help us visualize functions. Using a graph, we can see how output changes when input changes. However, visualizing complex functions is tricky.

As an example, let's take the complex function $f(x) = x^2 + 1$, where x is a complex input. It's a second degree polynomial, and hence, has two roots. However, f does not have real roots; its roots are given by $x = \pm\sqrt{-1}$. In other words, the roots of $f(x)$ are:

$$x = i \quad \text{and } x = -i$$

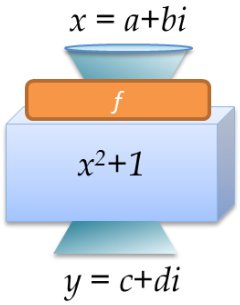
Consequently, we can write f as a product of its two linear factors:

$$f(x) = (x + i)(x - i)$$

What does it really mean to have i or $-i$ as a root? It means that if we use i or $-i$ as an input, the output must be zero. We can easily visualize real roots with the graph of a function. The roots are x-values, where the function crosses the x-axis. But how on earth are we going to visualize a graph crossing at i and $-i$?

In order to visualize imaginary roots, let's try to graph this complex function. For a complex function, the input and the output are both complex. Thus, the input variable x is a complex variable, which has the form:

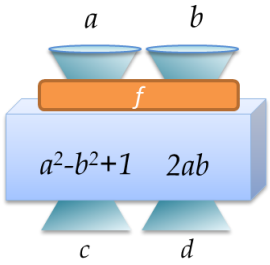
$$x = a + bi$$



where both a and b are real numbers. Now, the output of a complex function is also complex. Let's represent this using y, which we will represent as $c + di$, as shown in the Visual Model. As per its recipe, function f just squares its input and adds one.

The graph of f is given below. Note that if we wanted to represent f with one graph, we would need a 4D graph. Have you ever drawn a 4D graph? Me neither. Hence, we use two 3D graphs to show c and d separately. Remember, c and d taken together, as $c + di$, represent the complex output. Therefore, we show the real component (c) of the output with one graph and the imaginary component (d) of the output with the other graph.

In order to draw the two graphs, let's see how we obtain each component of the output. We know that:



$$\begin{aligned} y &= x^2 + 1 \\ &= (a + bi)^2 + 1 && [\text{ since } x = a + bi \,] \\ &= a^2 + 2abi + (bi)^2 + 1 \\ &= a^2 + 2abi - b^2 + 1 && [\text{ since } (bi)^2 = -b^2 \,] \end{aligned}$$

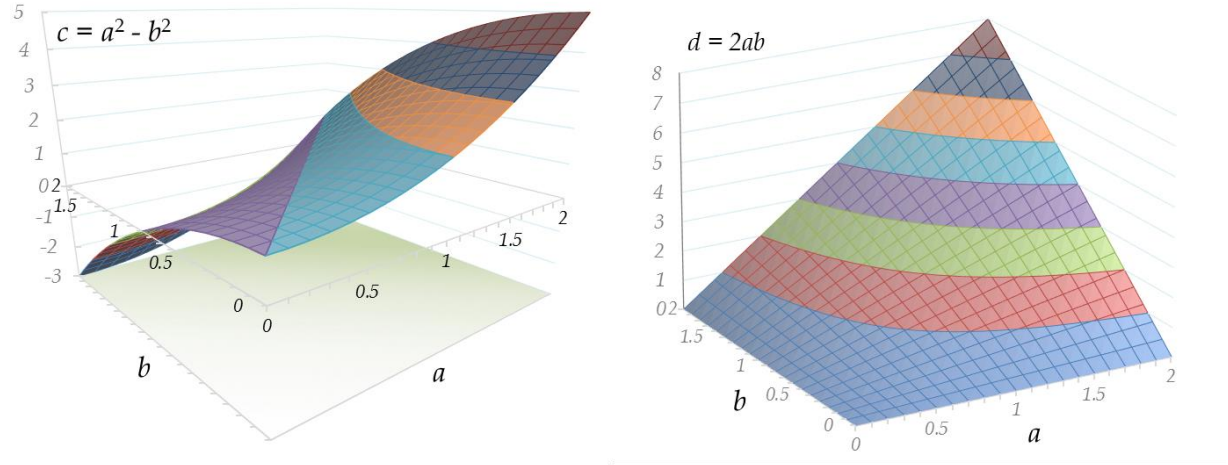
Separating real parts and imaginary parts, we get:

$$c = a^2 - b^2 + 1 \quad [\text{ real part }] \quad \textbf{(1)}$$
$$d = 2ab \quad [\text{ imaginary part }] \quad \textbf{(2)}$$

To summarize, a complex function accepts one complex number as its input and produces one complex number as its output. However, we can look at each complex number as a combination of two parts, real and imaginary. The second Visual Model summarizes these two relationships. Recall that a function cannot produce two outputs, but, we can model each *output component* as a separate function. That's exactly what we do when we graph a complex function — we produce two graphs, one for each *output component*.

Note that the graphs show only the 1st quadrant (i.e., when both a and b are positive). For any given input values a and b, you can read the real output, c, from the left graph and the imaginary output, d, from the right graph.

To start with, what's the output when the input is zero —i.e., when both a and b are 0? From the left graph, we get c=1; from the right graph, we get d=0. This is expected because $f(0) = 1$.



Now, let's look at the real component, c, of the output, from the left graph. The real component, c depends on both a and b. In order to provide purely real inputs, if we keep b at 0 and increase a, the output value c rises sharply (output is the square of input).

To provide purely imaginary input, let's keep a at zero this time and change b. As we increase b, c falls sharply. When b=1 (i.e., x=i) , the real component of output, c, is zero.

How about the imaginary component, d, of the output? From the graph on the right, we can see that along the axis labeled b, the output d is zero. Therefore, at x=i, both c and d are zero, which makes x=i a root of the function $f(x)$. Whoa! We found a root! In other words, if we use i as the input to function $f(x)$, the output of the function is zero. This is what we mean when we say a function has a non-real root.

Take a moment to study both graphs carefully. As a quick exercise, can you explain how to find the output value for input 1+2i using the two graphs? For input value 1+2i, both the real and imaginary components of the output are nonzero.

If we wanted to automate finding output, we could use the computer code given below. It calculates the real and imaginary parts of the output using two separate functions. We can produce output values for any input using this code to plot the two graphs we saw above.

```
function real_out( real_in, imag_in )
{
      result = real_in * real_in - imag_in*imag_in + 1
      return result
}
function imag_out( real_in, imag_in )
{
      result = 2 * real_in * imag_in
      return result
}
```

```
c = real_out( 1, 2 )     // calculate real part of output
d = imag_out( 1, 2 )     // calculate imaginary part of output
```

6.5 Complex Numbers as "Complete" Numbers

At first glance, you may think that complex numbers are rather bizarre. You may think that real numbers are more "real" than complex numbers. That's not the case.

What's the use of numbers? Numbers are used to represent real world objects. Real numbers are inadequate for doing that. That's what we saw with our cannon puzzler. That's what we see algebraically with $\sqrt{-1}$. Real numbers are only half of the story. Initially, we thought that whole numbers were enough to represent real-world objects. We found out that was not the case. Then, we thought that real numbers were sufficient. Again, we were proven wrong. Complex numbers are the "real deal". That's because they can represent more objects in the real world than so-called real numbers. In other words, complex numbers are a more powerful abstraction than real numbers. At the same time, when we apply basic algebraic operations to complex numbers, the result is always a complex number (which includes both real and imaginary numbers). Therefore, complex-numbers are "complete".

In many real-world situations, we can ignore the rotation of objects. You don't need to consider the rotation of apples when you count how many you ate. However, there are many other situations where rotation is a natural behavior. For example, think about a motor or a generator. By their very nature, they rotate. To model such objects and the properties associated with them, we need complex numbers.

Let's look at a concrete example from electrical engineering. Earlier, we looked at the relationship between voltage and current, when current is flowing through a resistor. To model this, we use the formula V = IR, which is called the Ohm's Law. Electrical circuits have two other types of elements that offer "resistance" to the flow of current. One is an

inductor, or a coil of wire as you often find in motors, that stores energy in a magnetic field. The other is a capacitor, which acts as a "reservoir" that stores electrical charge. Both inductors and capacitors offer a form of "resistance" to the flow of current. This resistance is known as "impedance", as in you *impede* someone's progress — in this case, inductors and capacitors "impede" the flow of current. However, impedance applies only to *change* of current flow. For instance, if the same amount of current flows through a wire continuously, the current will not see any impedance due to an inductor. However, once the current starts changing, the current will experience an "impedance" which affects the amount of current that can flow. As an example, if the current changes as a sine wave, an inductor will "impede" its flow. This impedance through an inductor is modelled using a modified version of Ohm's law:

$$\text{Voltage} = \text{Impedance} * \text{Current}$$

Which is similar to

$$\text{Voltage} = \text{Resistance} * \text{Current}$$

Now, you may wonder, why we are so interested in sine waves. If you understood the Fourier series in Section 5.4, you already know the answer. We can represent any waveform as a series (a linear combination) of sine waves. That's why we are so interested in sine waves. If we know how an electrical component reacts to sine waves, we can figure out how it reacts to *any* waveform.

As we saw in Chapter 4, a general sine wave has an amplitude, frequency, and a phase shift. Remember the Ferris wheel example from Section 4.6 on trigonometric functions? If the Ferris wheel with radius of 1 unit rotates 2 times a second, its frequency, f, is 2 (angular frequency, ω, is 4π). And if your friend sits a quarter circle ahead of you, she has a phase shift of 90^0 with respect to you. That is, she "leads" you by 90^0, or you "lag" her by 90^0. So, at a given time t, your height (from the hub of the Ferris wheel as we did in Section 4.6) can be represented with $\sin(4\pi t)$, and your friend's height can be represented with $\sin(4\pi t + \pi/2)$. If your mom sits on a Ferris wheel that has a diameter three times as large as yours, but she sits at the same angle as your friend, her height is given by $3\sin(4\pi t + \pi/2)$.

For a sine wave with an angular frequency of ω, the impedance of an inductor (i.e., a coil of wire) with an impedance L is given by:

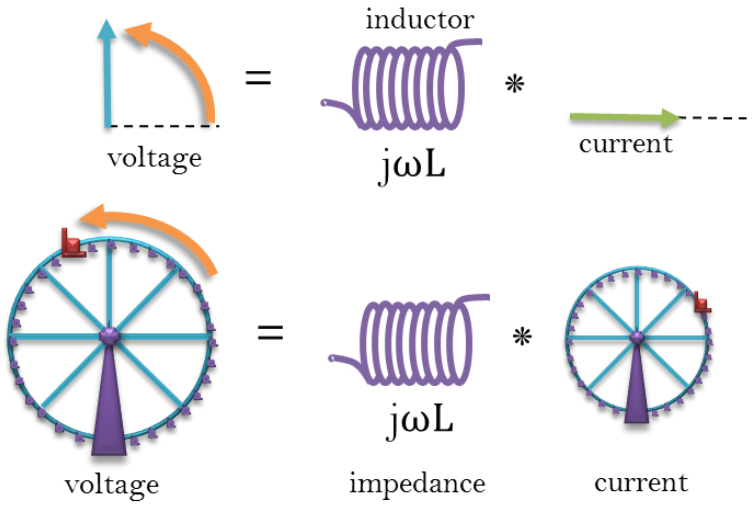
$$\text{impedance} = j\omega L$$

Note that in electrical engineering, we often use the letter j instead of i to represent an imaginary number, because i represents instantaneous current. Remember, it doesn't matter what letter we use to represent the imaginary unit. It still does the same thing.

What does jωL mean? First of all, the inductance, L, is just a scalar value. The larger the coil, the larger the value of L. It is analogous to the value of resistance. It's just how big the inductance is. The larger the inductance, the larger the impedance.

Second, the impedance depends on the frequency of the sine wave. If we double the frequency of the sine wave (if the Ferris wheel goes 2 times as fast), the impedance doubles. If the frequency is zero (i.e., for direct current), the inductor appears as a short circuit, without offering any impedance to the flow of direct current. Note that this is not the case with a resistor. The resistance does not depend on the frequency of current.

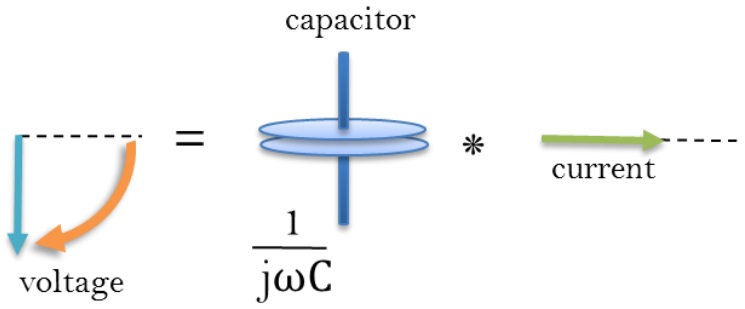
Third, and most importantly, we have a j stuck in front of ωL. What does that mean? It means that the impedance, unlike resistance, is a complex (rotated) object. What does it do exactly? Well, if we represent a sine wave pictorially as an unrotated arrow (pointing along the real axis), a sine wave that "leads" by 90^0 can be represented by another arrow with a rotated by 90^0 counter-clockwise, as shown below. These diagrams are referred to as phasor diagrams, because they show the relative phase difference between input and output sine waves.



You can also visualize this using our Ferris wheel model, which we use to represent a sine wave. If we have a rotating Ferris wheel representing the input sine wave (current), the output is another Ferris wheel, rotating 90^0 ahead, representing voltage. That is, the output is "leading" the input by 90^0. That is, if you were sitting at the 2 O'clock position on the input Ferris wheel, on the output Ferris wheel, you would be at 11 O'clock position.

Why does an inductor, which is just a coil of wire, do this? It's because when current starts increasing through an inductor, a magnetic field starts building up, resisting (or rather, impeding) the flow of current. For a pure resistor, there is no such magnetic field, and hence does not produce any phase shift in output.

In contrast, the impedance of a capacitor is modeled as $1/j\omega C$, where C is the capacitance, which represents how much "capacity" there is to store current. C is just a scalar value. As capacitance increases, the impedance drops. That is, if we have a larger "reservoir", it can pump (and sink) more current. In a capacitor, as frequency increases, the impedance drops, which is why ω is in the denominator. If the frequency is zero (i.e., for direct current), the impedance is infinite, and the capacitor appears as an "open circuit", where current cannot pass through. Note that 1/j is same as -j, as you can see easily by multiplying both the denominator and numerator by j. Thus, the capacitor rotates its input current by -90^0, to produce the output voltage, as shown below. In other words, it causes the voltage to lag 90^0 behind the current. Can you visualize this using our Ferris wheel analogy?



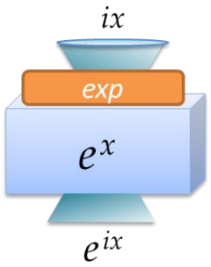
Again, why did we look at all this? Well, I wanted to show you a typical practical use of complex numbers. Complex numbers can be used to build models that perform rotation. Electrical circuits are built using resistors, capacitors, and inductors. As such, complex numbers are an absolutely essential tool if you want to analyze electrical circuits.

6.6 Non-real Exponents (Advanced Topic)

This section covers an advanced topic. You may skip this section but this section shows you the real power of complex numbers and why they are a fundamental tool in many disciplines of engineering.

When we talked about exponentiation above, we limited the discussion to real exponents. However, one of the most useful and surprising functions arise when we use imaginary or complex exponents. So, let's take a look.

6.6.1 Exponentiation with an Imaginary Input



As you may remember, we looked at the (natural) exponential function e^x, in Section 4.3.1. What if the input to the exponential model (i.e., the exponent) is imaginary? You could represent such an exponential function as:

$$f(x) = e^{ix}$$

where x is a *real* number (so ix is an imaginary number). What in the world does this mean? Well, when the exponent is an integer (say n), we multiply the base e by itself n times. But how do we multiply e by itself ix times? We have no clue how to do that.

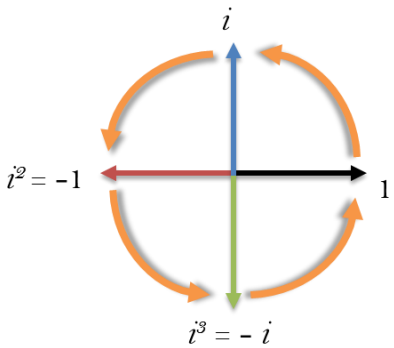
Fortunately, we have another definition of e^x. In Section 5.3, we expressed e^x as an infinite series of powers of x (called a power series). Recall that this series is like a polynomial with an infinite number of terms, where the terms get smaller and smaller as we go on. That power series expansion of e^x is shown below:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

We use this power series to calculate e^x (e.g. $e^{2.305}$), especially when x is a fraction. However, the above series is for a real exponent. So, instead of x, let's substitute ix, to find e^{ix} as:

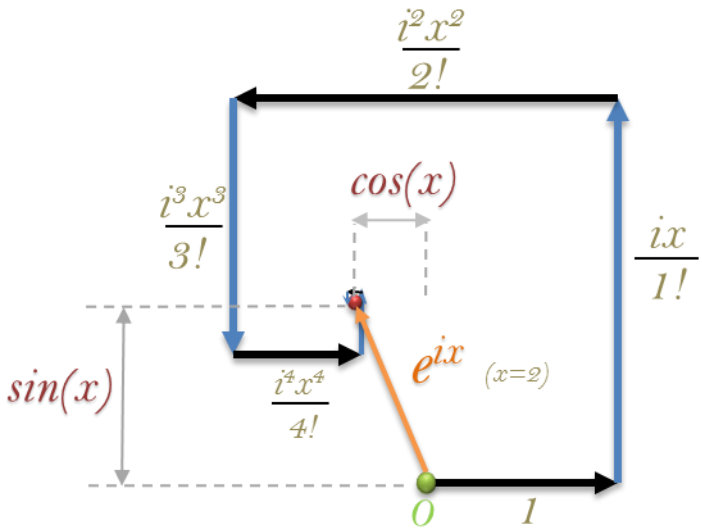
$$e^{ix} = 1 + \frac{ix}{1!} + \frac{(ix)^2}{2!} + \frac{(ix)^3}{3!} + \frac{(ix)^4}{4!} + \cdots$$

Note that the numerator (top part) of term n is $(ix)^n$, which is $i^n x^n$. We have already seen i^n before. Remember the rotating unit circle? It is shown again here to refresh your memory.



Now, let's look at the sum of this series. There are two ways to look at this. The first way is pictorially, as shown below. Let's start at zero and keep on adding each term of e^{ix}. First, we add the 0th term, which is 1. Then, we add the 1st term, $ix/1!$, to it. Then, we add the 2nd term, $(ix)^2/2!$, to it. Note that $(ix)^2$ is $-1x$, because $i^2 = -1$. Therefore, the 2nd term is $-x/2! = -x/2$. Similarly, the numerator of the 3rd

term is $(ix)^3$, which is equal to $-ix$, if we substitute -1 for i^2. As you can see, each new term is 90^0 rotated from the previous term.



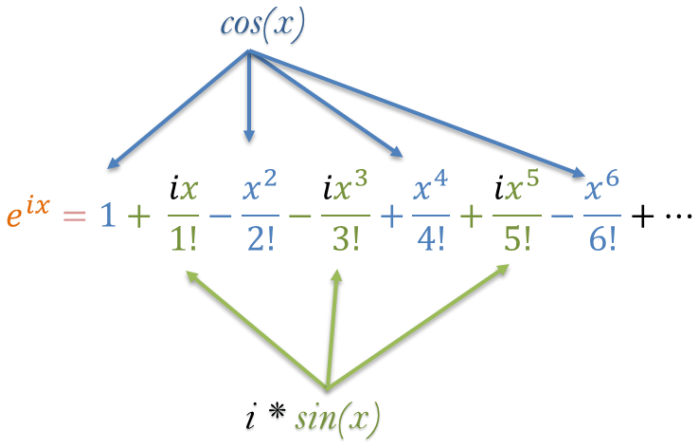
What about the magnitude of each term? As we already know, the numerator of each term is a power function, while the denominator is a factorial function. We know that the factorial function grows much faster than the power function. Thus, as n increases, the denominator becomes much larger than the numerator, making the magnitude of each term smaller and smaller, really fast. Effectively, we are adding ever diminishing rotating terms to create some sort of a "spiral". The figure above shows how we can obtain the value for e^{ix} when x=2 (i.e., e^{2i}), using this method. Each arrow shows each term, starting at 0.

Now, let's look at the end point of this "spiral". Here is the real shocker. The real value of this point is cosine of x and the imaginary value of this point is sine of x. For instance, when x=2, the real value gives cos(2) and imaginary value gives sin(2).

To understand why, let's look at this in the algebraic form. Let's start with the power series expansion of e^{ix}, with -1 substituted for i^2 whenever possible. That series is shown below.

$$e^{ix} = 1 + \frac{ix}{1!} - \frac{x^2}{2!} - \frac{ix^3}{3!} + \frac{x^4}{4!} + \frac{ix^5}{5!} - \frac{x^6}{6!} + \cdots$$

Now, let's group the terms of e^{ix}, so that all real (blue) terms are together and imaginary (green) terms are grouped together, as shown below. Further, from the imaginary terms, let's factor out i, because i is common. Do you see what we get? All of the real (blue) terms constitute the power series expansion of the cosine function, as we saw in Section 5.3. Similarly, all the imaginary (green) terms constitute the power series expansion of the sine function as we saw in Section 5.3!



This is a really fascinating and hugely important result. This is one of the benefits of studying the power series: to see how everything is connected. With power series, we looked at functions as an infinite series (sum) of terms. This helps us see the connection between functions of *complex* variables (e^{ix}) and functions of *real* variables, $\cos(x)$ and $\sin(x)$. In other words, a function of a complex variable can represent *two functions* of real variables. This shows us the power of complex numbers and why we study them.

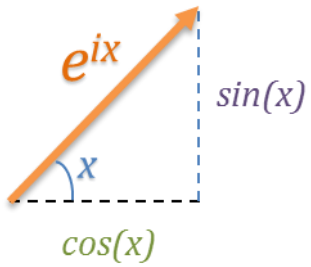
The result we obtained above is called the Euler's Formula, or the **Euler's Identity**. It can be expressed as:

$$e^{ix} = \cos(x) + i \sin(x)$$

Let's take a moment to appreciate what this identity is saying.

It says that e^{ix} can simultaneously represent two functions — one sine function and one cosine function! Let's further visualize this with the aid of our Ferris wheel analogy. This says that with e^{ix}, we can simultaneously model your *height* and your *horizontal distance* on the Ferris wheel using *one* function. In other words, e^{ix} completely models your position on the Ferris wheel. For a given value of x, the function e^{ix} gives your exact position (both horizontal and vertical) on the Ferris wheel. As x varies, e^{ix} completely describes your rotation (in a circle). Notice that x is real, so if we use x to represent time, e^{ix} models your position on the Ferris wheel as time goes on.

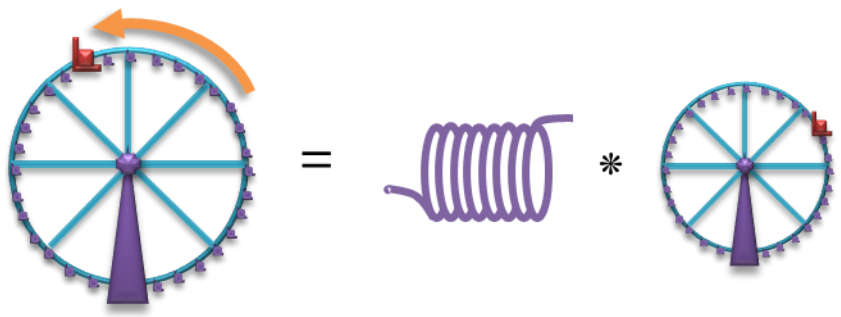
Notice that the right-hand side of the Euler's Identity consists of a real part and a imaginary part, just like a complex number, $a+bi$. Consequently, we can look at e^{ix} as a complex number that makes an angle x with the real axis, as shown in the figure. It can simultaneously represent both the sine value of x and cosine value of x. As x varies, e^{ix} traverses a circle of radius 1. Instead of a radius of 1, if we have a radius of r, then re^{ix}, can represent any complex number with a magnitude of r and an angle of x. This is known as the **polar representation** of a complex number, giving us the identity:



$$a+bi = re^{ix} \qquad \text{where} \qquad a = r\cos(x) \quad \text{and} \quad b = r\sin(x)$$

You may ask what the big deal with e^{ix} is, since the sine and cosine functions could do the same. True, but not completely. A sine or cosine function is only half as good as e^{ix}. For a given value of x, the sine function can describe only your vertical position (height) on the Ferris wheel. Similarly, the cosine function can describe only your horizontal position. The function e^{ix} is twice as useful as sine or cosine alone. It can model *both* your vertical and horizontal position *simultaneously*, as you rotate on the Ferris wheel. Because of this, we often use e^{ix} to model rotation and to represent sine and cosine functions.

For instance, to model rotation with angular frequency ω and amplitude A, we can just use $Ae^{i\omega t}$, where t is the input variable representing time. Both ω and A are constants. The function $Ae^{i\omega t}$ represents both $A\sin(\omega t)$ and $A\cos(\omega t)$ functions. Here is a simple example. In Section 6.5, in our phasor diagram, we used line segments to pictorially depict an input wave and an output wave. Now, we have a full-fledged function that can represent a sine or cosine wave. At any given point in time, t, we can represent the relationship between current through an inductor and voltage across it as:



Let's look at the above statement closely with our Ferris wheel analogy. The above says that if we use a Ferris wheel with a diameter A, rotating at an angular frequency ω, we will get an output Ferris wheel rotating at the same frequency ω, but with a different diameter ($A\omega L$) and rotating 90^0 ahead. If you think in terms of sine or cosine waves, it describes the same relationships. If we have an input sine wave with angular frequency ω and amplitude A, we will get an output sine wave with amplitude multiplied by $A\omega L$ and phase shift of 90^0.

Due to the versatility of $e^{i\omega t}$ in representing rotation, it shows up in many electrical engineering and signal processing applications. For instance, you will find $e^{i\omega t}$ in Fourier Transforms.

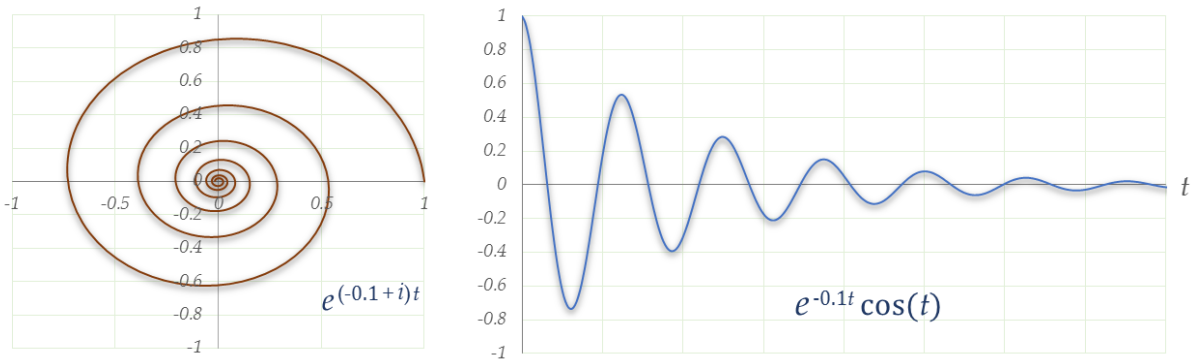
6.6.2 Exponentiation with a Complex Input

Instead of having just an imaginary value in the exponent, what if we had a whole complex number? We can represent such a function as e^{st}, where s is the complex number $a+bi$ as shown below:

$$
\begin{aligned}
e^{st} &= e^{(a+bi)\,t} \\
&= e^{(at+ibt)} \\
&= e^{at} * e^{ibt} && \text{[adding exponents is multiplication: e.g. } e^{a+b} = e^{a} * e^{b} \text{]} \\
&= Ae^{ibt}, && \text{where } A = e^{at}
\end{aligned}
$$

What does this mean? Well, we have a product of two functions. The first, e^{at} is the familiar natural exponential function similar to e^x. The second, e^{ibt}, is similar to e^{ix}, representing rotation (a Ferris wheel). Note that both a and b are constants, but the expressions at and bt

are variables because t is the input variable. Let's take the case where a is negative. Then, e^{at} is the exponential decay function, which acts as the amplitude A for the waveform e^{ibt}. Since e^{at} depends on time, the amplitude A changes with time — in fact, A decays exponentially as time goes by. Thus, when a is negative, e^{st} represents a Ferris wheel, whose radius decreases rapidly (exponentially) as time goes by. In other words, e^{st} models a spiral whose radius is decreasing exponentially. The constant a decides how fast the output decays. The graph of this Ferris wheel with an exponentially decreasing radius is shown on the left below, while the graph of the real component (i.e., horizontal distance to the hub) is shown on the right. For these graphs, we have picked the values a=-0.1 and b=1.



As you can see from the graph on the left, the radius of the Ferris wheel starts at 1. Thus, the point (1, 0) on the right is the starting point at t=0. As time goes by, the radius decreases exponentially, finally approaching zero. Remember, in this example a is negative. If a is positive, instead of the rapid decay, we would get rapid (exponential) growth of amplitude.

The graph on the right shows the real component of the Ferris wheel (i.e., the horizontal distance from the hub). It shows a cosine waveform whose amplitude is decaying exponentially. Therefore, if we were to look at the real component of e^{st}, we would get an exponentially decaying cosine wave. Similarly, if we considered the imaginary component of e^{st}, we would get an exponentially decaying sine wave.

You may wonder why we went into all this trouble to build this exponential model with a complex variable as input. The reason is the real-world importance of this model. Have you ever played a key on a piano or any other musical instrument and observed how the sound dies down as time goes by? This is the model representing that. As you can see, this model captures both sine and cosine waves that dies down (dampens) as time goes by. This model, e^{st}, is more general and is closer to nature than the model e^{it}, because the model e^{st} captures this natural dampening effect. Imagine a world where when you pluck a string of a

guitar, the sound never dies down. That would be a totally different world from what we live in.

Therefore, not surprisingly, this e^{st} model is used in many branches of science and engineering. For instance, you will meet this model in a really useful tool called Laplace Transforms, which allows you to express any waveform as a combination of *decaying* sine waves. Compare this with Fourier Transforms, where any waveform is expressed as a combination of non-decaying sine waves. Therefore, Fourier Transforms are a special case of Laplace Transforms with a=0.

Isn't it truly fascinating how complex numbers are at the heart of representing such a rich set of mathematical models? Don't forget that fact next time you ride a Ferris wheel, which of course, won't have a decaying radius.

6.7 The Story So Far

In the previous chapters, we looked at members of the function dynasty, how they came into existence, their common properties, their common problems, and the objects they consume and produce. We also looked at the common models, and how an infinite series of simpler functions can model more sophisticated functions. In this chapter, we revisited objects that functions consume and produce, because functions showed us that real numbers are just half of the story.

Functions, in particular square root, showed that it could produce a new kind of output that is not a real number, even when using real numbers as input. These new numbers represent objects with a direction in 2D. That made us realize that any function can accept these "rotated" objects. In particular, we saw multiplication with a complex input performing rotation. We also found some of the "missing" roots of polynomials — they were just hiding from us because they were rotated, and earlier we did not have the capability to see rotated objects. That is, earlier, we did not use complex numbers as inputs to functions.

In Section 4.3, with real numbers as exponents, we saw that the exponential function produces rapid growth or decay. When using complex numbers as input, the same exponential function could play an additional role: it could produce a rotating output, representing both a sine and a cosine function at the same time, with constant, decaying, or growing amplitude.

A carpenter who has access to plywood or manufactured wood can produce a usable chest. But the same carpenter, given mahogany or ebony, can produce a chest that could become a masterpiece. Similarly, complex numbers representing rotated objects give the same old

functions the capability to produce much richer output. We saw how the functions and the objects they consume enhance each other's value.

We also saw that these rotated objects and functions that use and produce them are not a mere mathematical curiosity. They are a fundamental tool used in several engineering disciplines, especially in electrical engineering, due to their ability to represent waveforms and models of electrical components.

In the next chapter, we will see how functions can consume and produce even richer objects.

7 FUNCTIONS IN 3D SPACE

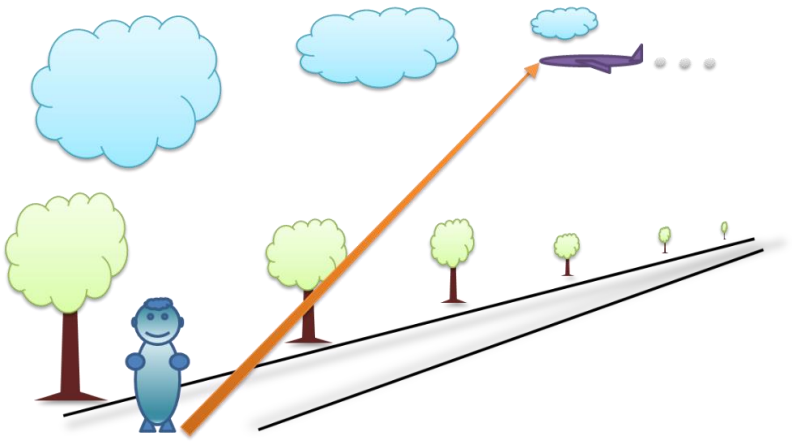
Chapter Overview: This chapter extends our journey of representing richer objects (and operators) to vectors, which can represent 3-dimensional or higher dimensional objects. First, we will look at vectors as a geometric concept and then as an algebraic concept, finally uniting them using the concept of a linear combination. In the process, we will look at new operators like the dot-product and the cross-product and richer relationships (functions) like scalar fields and vector fields that are indispensable in modeling the real world.

7.1 Vectors: Representing Objects in Space

If you asked me how to get to the nearest gas station, and if I replied "3 miles", you would, understandably, be baffled. Your very next question would be, "3 miles in which direction?" Now, if I said, "exactly 3 miles north from here", you should be able to find the gas station. The number "3 miles" is plainly insufficient to represent the relationship between where you are and the location of the gas station.

In Chapter 3, we saw that whole numbers are sufficient to represent how many apples one has, but insufficient to represent half an apple. We needed fractions and real numbers for that. Then, we saw how real numbers could not represent objects that are pointing in different directions on a two-dimensional (2D) plane. That led to complex numbers. Is that all the objects we want to represent?

As an example, if you see an airplane flying in the sky, how would you describe its position relative to where you are? The airplane may be flying one mile high, two miles east from where you are and a half a mile north of where you are. A whole or a real number can't represent that. Neither can a complex number.



Therefore, we need a new mathematical object that can represent the position of an object in space. We can call such an object, a **vector**. Note that, just like any other number, a vector is a representation, or an abstraction.

A vector can be geometrically represented as a directed line segment. That is, a line segment with an arrow at the tip, which points in the direction we want. The length of the line segment represents the magnitude of the vector. From the figure above, you can see how the vector (arrow in figure) can represent the relationship between your position and the position of the airplane. In other words, a vector can represent a relationship between two objects in three-dimensional (3D) space.

Note that a real number cannot express the above relationship. A real number can tell the distance between the two objects but cannot represent the relative direction. A complex number cannot represent directions in 3D space. If we had the objects on a 2D plane, a complex number could represent the relative position of one with respect to the other.

Note that for this chapter, we consider 3D space by default, and hence, our vectors are 3D. However, vectors can represent objects in 2D, 3D, or higher dimensions. As for terminology, real numbers that we dealt with so far are called **scalars**, because they cannot represent a direction.

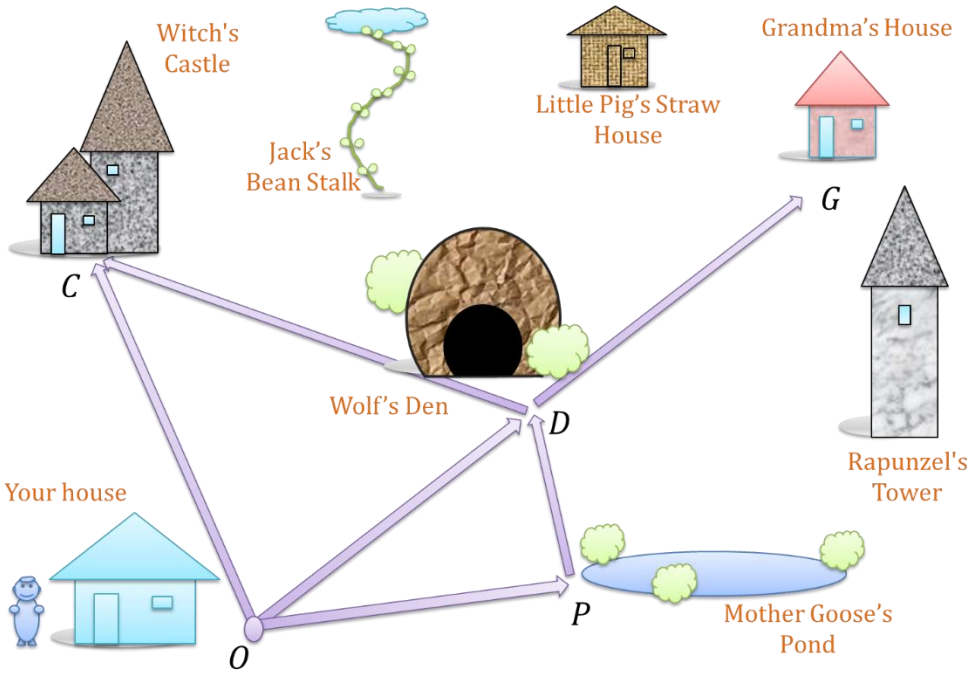
7.2 Where do Babies, err…, Vectors, Come From?

A number can represent the difference between two situations (states). For instance, when you have 3 apples, you compare it with a situation where you do not have any apples. The difference between those two situations is represented by number 3. Fundamentally, a number arises from a comparison, or a difference, between two states. So do vectors. They arise by comparing (finding difference between) two positions in space.

Imagine you lived in the fairytale land, as shown in the figure below. Your house is at the bottom left corner of the map, at point *O*. You can draw a bunch of vectors connecting places on the map. For instance, vector ***OP*** gives both the distance and direction to Mother Goose's Pond from your house, and vector ***PD*** gives the distance and direction to Wolf's Den from Mother Goose's Pond. As you can see, a given vector represents the difference between two places on the map. When we represent a vector by a line segment (e.g., ***OP***), the first letter (e.g., O) refers to the starting position and second letter (*P*) specifies the end position. It's just a way to give each vector a name. We always represent a vector with boldface to indicate that it is a vector quantity with both a magnitude and a direction (as opposed to scalars).

If you draw all of the vectors from your house (point *O*) to all other places, that would be sufficient to define all the places on the map. Why? Well, because, when you specify vector

OD (your house to Wolf's Den) and ***OP*** (Your house to Pond), that determines vector ***PD*** (Pond to Den) as well.



7.3 Vector Difference (Subtraction)

Relative to your house, vector ***OC*** represents the Witch's castle, and vector ***OD*** represents the Wolf's Den. So, what's the *difference* between those two places? From the map, it's vector **DC**, the path from Wolf's Den to the Witch's Castle. We can write this difference as:

$$\boldsymbol{OC} - \boldsymbol{OD} = \boldsymbol{DC}$$

There is another way to look at this. Imagine you start at your house (O) and your final destination is the Witch's Castle (C). However, first you walk along path ***OD*** to Wolf's Den (D). Now, you are at a halfway point. What's the remainder of your journey? The remaining section is given by ***DC***, which is given by the difference between your final destination and the halfway point, D. Notice that when you find the scalar difference 5 – 3, it means the same thing. If you already have 3 apples, how many more do you need to get to 5 apples? The answer is 2, which is the difference between the two numbers. Similarly, if you are already at point D, how far do you need to travel to get to your final destination C? That difference is given by vector ***DC***.

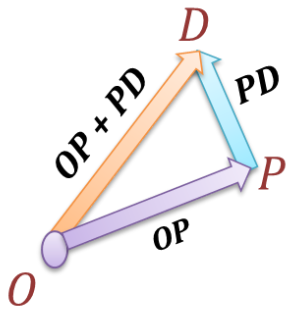
This is an absolutely important concept to grasp.

7.4 Vector Addition

Vector addition can be easily understood with our map as well. Let's start at your home. First, if you travel along vector ***OP*** to the Pond, and then go along vector ***PD*** to the Den, what's the final destination? It is D, which is represented by vector ***OD***. It is same as going from your home to the Den directly, along vector ***OD***. We can represent this as:

$$\boldsymbol{OP} + \boldsymbol{PD} = \boldsymbol{OD} \qquad \textbf{(1)}$$

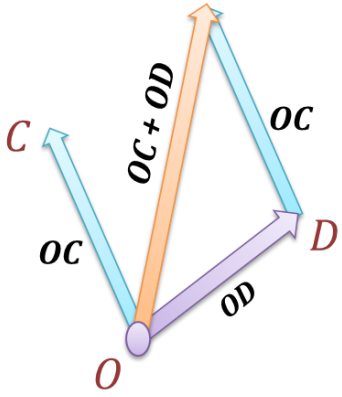
This is called the "**triangle rule**" for vector addition. From the map, you can see that vectors ***OP***, ***PD***, and ***OD*** are sides of a triangle. The triangle rule can help you with vector difference as well: if you subtracted ***OP*** from both sides of equation (1), you get:



$$\boldsymbol{PD} = \boldsymbol{OD} - \boldsymbol{OP}$$

This is a vector difference. Can you explain this vector difference using our map? You should be able to notice how vector difference and addition are related. Take vector ***OP***. If we add it to ***PD***, you get ***OD***. If we subtract it from ***OD***, you get ***PD***.

What if we wanted to add ***OC*** and ***OD*** together? In (1), ***PD*** started where ***OP*** ended (at point P), so we could add them easily. In that case, the tip of one vector was connected to the tail of the next vector. However, if you wanted to add ***OC*** and ***OD*** together, we have two options. Option 1 is to move ***OC*** so that ***OC*** starts where ***OD*** ends, as shown in the figure. Option 2 is to move ***OD*** so that ***OD*** starts where ***OC*** ends. Can you draw that diagram and verify that both options produce the same result?



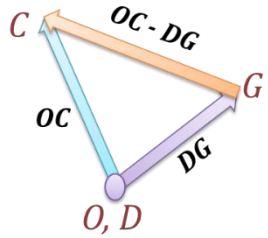
This demonstrates one very important aspect about a vector. A vector represents a length and a direction. That's it. You can "ground" it anywhere you want. There is no fixed starting position. In other words, a vector represents a *difference* between the positions of two objects. Therefore, it can represent the difference between *any pair of objects* that has the same difference.

You can "ground" a vector anywhere

Let's take our scalar example again. We know that the difference between 5 and 3 is 2. The number 2 represents a difference. It doesn't always have to be between 5 and 3. The

number 2 can represent the same difference between 12 and 10. Vector subtraction works in the same way with one caveat: vector difference represents both a magnitude (length) and a direction, whereas scalar difference represents only a magnitude. Other than that, both scalars and vectors can represent difference between any two objects.

As a result, when applying the "triangle rule" for vector addition, we are free to move any vector as we please. The same procedure applies to finding the difference between two vectors that do not have a point in common. For instance, if we had to find ***OC*** – ***DG***, we can move point D to coincide with O to calculate the vector difference.



7.5 Multiplication by a Scalar (Scaling)

On our fairytale land map, when you start from your house, the Grandma's house is in the same direction as the Wolf's Den. Isn't that that the reason why you can't visit grandma without the wolf finding out about it?

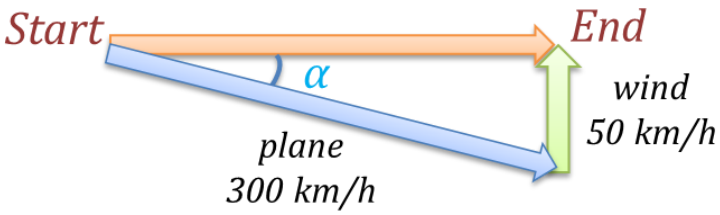
If we represent Grandma's house by vector ***OG*** and Wolf's Den by vector ***OD***, we can represent ***OG*** as a scalar multiple of ***OD***, as:

$$\boldsymbol{OG} = 2.2 * \boldsymbol{OD}$$

Here is what this means: if you have to go some distance to Wolf's Den, if you go 2.2 times that distance in the same direction, then you will end up at Grandma's. As you can see, scalar multiplication just makes the length (magnitude) of a vector larger or smaller without changing its direction.

7.6 Examples of Vector Addition and Subtraction

It was fun to see how you can find your way around in the fairytale land using vectors. However, unfortunately, we don't live in the fairytale land. Why do we need vectors in the real world? To understand the serious uses of vectors, you have to wait a little bit more, but for now, here is a very simple use. Say you are flying an airplane at 300 kilometers per hour. You want to go straight east. However, there is a wind blowing due north 50 kilometers per hour. In which direction should you point the airplane? Further, after one hour, what is your position with respect the position your started at?



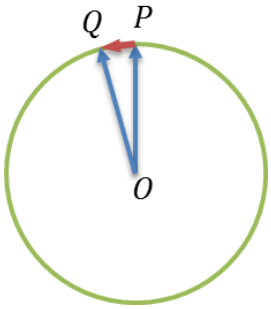
The above picture shows the vectors corresponding to this problem. If we want to end up at the right place, we have to travel along the blue arrow, because the wind is going to push us north. For instance, if we travel for one hour, the wind is going to take us 50 km north.

From the diagram, you see that $\sin(\alpha) = 50/300 = 1/6$. Thus, $\alpha = 9.6^0$. So, we have to travel 9.6^0 degrees clockwise from the east direction, to end up east. To find out where we would end up, we have to add the blue arrow and the green arrow together, resulting in the orange arrow. You can find the magnitude of the orange arrow using the Pythagorean theorem or trigonometry.

Instead of the velocity of the wind, if we were given the velocity of the airplane (blue arrow) and where we would end up (i.e., the orange arrow), then we can find the velocity of the wind (green arrow) using vector subtraction. That is, we have to subtract the blue vector from the orange vector.

Although vectors gave us insight into solving this problem, we had to resort to geometry (Pythagorean theorem) and trigonometry to find the actual magnitudes and angles of our flight path. We will address this deficiency soon, once we have an algebraic representation of vectors.

Vectors are used in navigation, both in air and on sea. Notice that we had to resort to vectors because the underlying physical quantities have directions. The velocity of the airplane and the wind *need* vectors. I made the above example easier by making it a 2D problem. In practice, this is a 3D problem where the wind could be blowing in any direction in 3D space. That's where vector algebra is really going to shine.



Another example of vector difference is shown in the figure. Here, we have an object rotating in a circle and we want to find its velocity (both magnitude and direction). Assume that at time t=0, the object is at point P. Therefore, vector ***OP*** represents the object's position at time t=0, with respect to O. After 1 second, the object is at Q. Therefore, the vector ***OQ*** represents the object's position at time t=1. Now, we want to find the velocity of the object. The average velocity of an object is the displacement divided by the time duration it

took to travel that distance. What's displacement? Displacement is the vector difference between two positions. In fact, in our fairytale land, all of the vectors we calculated were displacements. So, what's the vector difference between ***OQ*** and ***OP***? It's the vector ***PQ***, which is shown by the red arrow in our diagram. Thus, within one second, we had a displacement shown by the red arrow. Since displacement divided by the time duration (1 second in this case) is the average velocity, the red arrow also represents the average velocity in this case. Notice the direction of this velocity. It is "fairly close" to the direction of the tangent at point P. In fact, if we found the velocity when Q was extremely close to P, the direction of that velocity would be same as the direction of tangent at point P. That's called the instantaneous velocity at P.

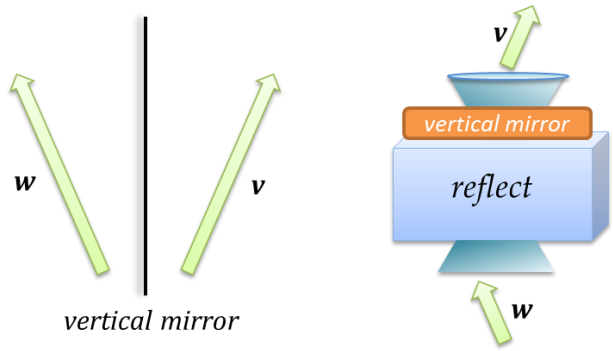
You can observe this in practice quite easily. If you tie a rock at one end of a string, hold it from the other end and swing the rock in a circle above your head, the rock will go in a circle similar to the one shown in the figure above. At some point in time, the rock will be at point P and you will be holding the other end of the string at point O. If you suddenly let go of the string, where would the rock go? Usually, through a neighbor's window!

Seriously, though, the rock, or any other moving object for that matter, travels in the direction of its velocity. In other words, the rock will travel along a line tangent to point P, assuming you release the string when the rock is at point P. The same thing happens when a car skids off the road while taking a sharp turn. If the car is travelling along the circle given in the above figure, and if it suddenly skids at point P, it would travel on the line tangent to point P, and end up in a ditch.

This example shows how vector representation and vector difference can illuminate real world observations we make.

7.7 Vector Functions You Meet Every Day

The same way functions accept scalars as inputs and produce scalars as outputs, functions can accept vectors as inputs and produce vectors as outputs. One obvious example is reflection. Reflection is a function, as we saw in Section 1.4. It can accept a vector as its input. The output is the reflected vector. The function "reflect" and its Visual Model are given below. Notice that we are representing a vector with a single bold letter below, instead of two letters we used above for line segments. Again, it's just a way of naming.

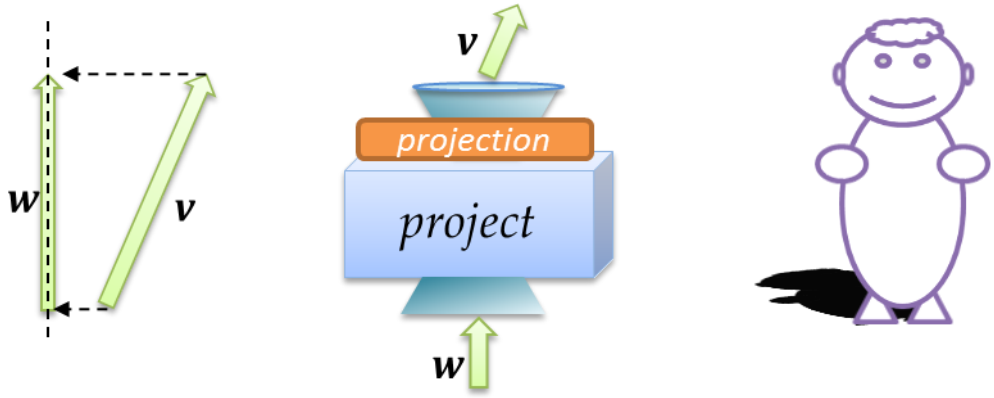


In textual form, we can write this function as:

$$\boldsymbol{w} = \text{reflect}(\boldsymbol{v})$$

where $\boldsymbol{v}$ is the input vector and $\boldsymbol{w}$ is the output vector.

Similarly, projection is a function. It projects a vector onto a plane. In the following figure, we project the input vector $\boldsymbol{v}$ onto a vertical plane, producing the output vector $\boldsymbol{w}$. The Visual Model for the projection function is also shown.



In textual form, we can write the "project" function similar to the "reflect" function, using the input vector $\boldsymbol{v}$ and the output vector $\boldsymbol{w}$, as follows:

$$\boldsymbol{w} = \text{project}(\boldsymbol{v})$$

As a practical example, notice that your shadow is created by a projection. As another example, the vector version of Newton's second law of motion,

$$\boldsymbol{F} = m\boldsymbol{a},$$

represents a function between two vectors, where the input vector $\boldsymbol{a}$ is simply multiplied by a scalar, m, to produce the vector output $\boldsymbol{F}$. Every time you move an object, you are

applying this law. The vector function captures that acceleration $\boldsymbol{a}$ and Force $\boldsymbol{F}$ have the same direction. If you push a lawnmower eastward, it goes eastwards. Duh!

7.8 Product Between Two Vectors

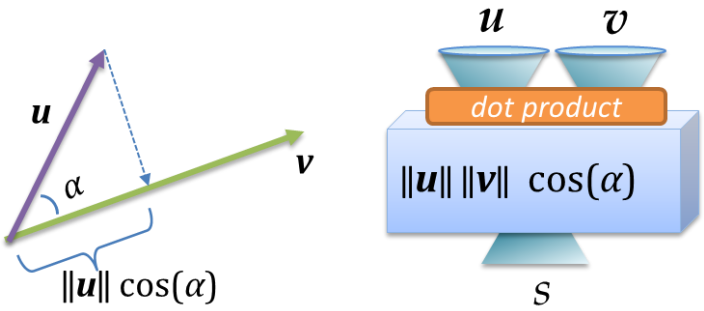
In Section 7.5, we looked at multiplying a vector by a scalar, to make the vector larger or smaller. There are two other types of products defined *between* vectors. Remember that a product is a function (an operator). We define a function to model some real-world relationship. So, with these two products, we define new functions (operators) to model some relationships that are useful to us.

7.8.1 Dot Product (Scalar Product)

The first product we define is called the **dot product**, or **scalar product**, between two vectors. The dot product between vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ is denoted by $\boldsymbol{u} \cdot \boldsymbol{v}$. It is defined as the product of the magnitudes (lengths) of the two vectors multiplied by the cosine of the angle between them. Why do we define it like this? In order to answer that question, let's look at what this definition actually says.

$$\boldsymbol{u} \cdot \boldsymbol{v} = \|\boldsymbol{u}\| \, \|\boldsymbol{v}\| \cos(\alpha)$$

where $\|\boldsymbol{u}\|$ represents the magnitude (length) of vector $\boldsymbol{u}$. Since the product is between magnitudes, which are scalars, and the cosine of the angle is also a scalar, the output of the dot product operation is a scalar, not a vector.



The figure above shows vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, and the angle α between them. It also shows the Visual Model of the dot product. Again, remember that output of $\boldsymbol{u} \cdot \boldsymbol{v}$ is a scalar, *s*. Hence, the dot product is also referred to as the **scalar product**.

What does the output of the dot product signify? To understand that, take a look at the above figure. Take a look at vector $\boldsymbol{u}$. If vector $\boldsymbol{u}$ is projected onto $\boldsymbol{v}$, we would get $\|\boldsymbol{u}\|\cos(\alpha)$, as shown on the figure. Again, it's a scalar value. Its length is shown by the curly

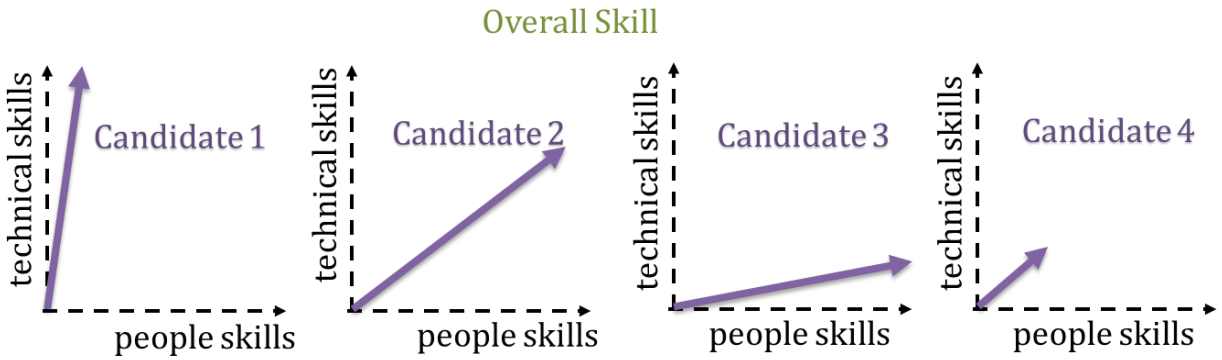
brace. The output of the dot product is this projection of $\boldsymbol{u}$ multiplied by the length of vector $\boldsymbol{v}$. So, we can write:

$\boldsymbol{u} \cdot \boldsymbol{v}$ = (projection of $\boldsymbol{u}$ onto $\boldsymbol{v}$) $*$ (length of $\boldsymbol{v}$)

Let's examine this a little bit closely. What happens to projection of $\boldsymbol{u}$, when $\boldsymbol{u}$ is large? The length of the projection increases. Similarly, what happens to the projection when the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ is small? The projection becomes larger as well, because cosine of an angle is greatest when the angle is zero. When $\boldsymbol{u}$ and $\boldsymbol{v}$ are perpendicular to each other, this projection is zero. In addition, as $\boldsymbol{v}$ increases, the output of the dot product increases. Thus, the output of the dot product becomes larger, as

(1) $\boldsymbol{u}$ gets larger
(2) $\boldsymbol{v}$ gets larger
(3) the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ gets smaller

Let's look at a real-world example to get an intuitive understanding. Say an employer wants to hire a bunch of sales people to fill some open positions in her company. All of these positions require two skills: technical skills (e.g., computer skills) and people skills (e.g., communication skills with customers to sell products). The employer is currently considering 4 candidates with the following skill profiles.



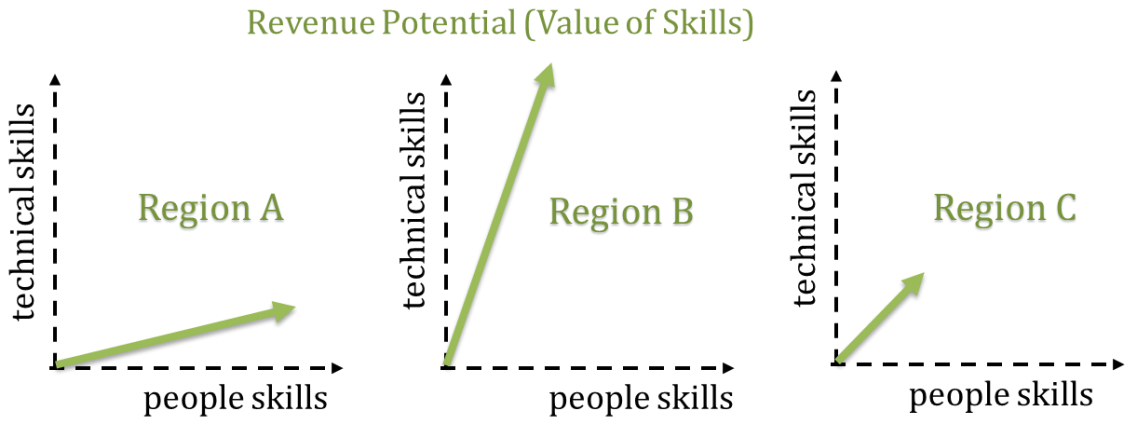
The overall skill of each candidate is represented by a vector because we consider two sperate skills as one. Assume that you can measure each skill separately and they are completely independent. For instance, you can give someone a technical test of 100 questions to measure his or her technical ability. Similarly, you can give a field test to measure "people skills" needed for the sales job — for instance, how many bags of dogfood a salesperson can sell — to people who don't have any dogs!

"Oh, you don't have any dogs? But, do you know someone in your family who has a dog? Oh, your grandma has a dog? Good for her! You know, dogfood is a great *gift idea*. You are going

to buy a Christmas present for your grandma, aren't you? Um, … have you thought about what you are going to buy for her dog? You know, I think she LOVES her dog, and she may be quite disappointed if you forget about her dog. I think you shouldn't disappoint your poor grandma. Besides, who knows, she might be considering making a contribution to your college fund … Oh, you want to buy 5 bags of dog food …. I knew you were a smart guy! I can just tell".

That's how you measure people skills. Once you have measured both skills, you can represent two skills using a vector. The length of the vector shows the total amount of skills the candidate has and the direction shows which of the two skills that the candidate has more of. For instance, Candidate 1 has a lot of technical skills but not much people skills. Candidate 3 has a lot of people skills but not much technical skills. Candidate 4 only has a small amount of each skill.

These candidates will be assigned to three sales regions. However, each region is different. In some regions, if you have good technical skills, you can make more sales and bring in more revenue (income). So, having technical skills would be more valuable for that region. It could be due to the product mix, the customer base, or some other factor. Similarly, in another region, it could be valuable to have more people skills. The revenue potential for each of the three regions is given below. For instance, if you have more people skills, in Region A, you can make more sales (and revenue). Technical skills are not very valuable there. Compared to the other two regions, it is harder to sell stuff in Region C. For instance, it is harder to sell snow tires in Hawaii, even if you are a really good salesman.



The employer wants to maximize the revenue generated from each region. For that, she has to decide which candidates to hire and which candidate to send to each region. How should she approach this?

You guessed right! We have to take the dot product between the "skill vector" and the "revenue potential" vector for each candidate, for each region. For instance, if you take dot product between Candidate 1 and Region 1, you get the revenue Candidate 1 can earn in Region 1. It doesn't take an MBA to realize that Candidate 1 will make more sales and earn more in Region B. Similarly, Candidate 3 will do well in Region A. For Region C, which is the most challenging region to sell stuff, Candidate 4 will do far worse than Candidate 2, because Candidate 2, who has more skills, can overcome some of the obstacles and make more revenue.

The above example should give you an intuitive feeling about what the dot product does. When a candidate and a region are "aligned", the output of the dot product is high. Otherwise, it is low. In real-world situations similar to our example, vectors will have many more dimensions (not just 2). Taking the dot product is one way to score (evaluate) a match between two objects. Next time you are on a date, you can try scoring your relationship with a dot product. You will see instant results. I wouldn't guarantee it would be pretty, though.

In physics, the dot product is used to calculate the work done by a force. When a force acts upon an object, it moves the object. The work done by that force is the dot product between the force and the displacement.

Work = **Force** · **Displacement**

Note that both force and displacement are vectors, because each has a direction. Why do we take the dot product in this case? The dot product says the following: what really counts as work is the displacement we get in the direction we apply the force. Isn't that obvious? If you push your lawn mower forward to mow grass, but it goes sideways on to the sidewalk because of a slope, you didn't get any real work done — you didn't mow any grass.

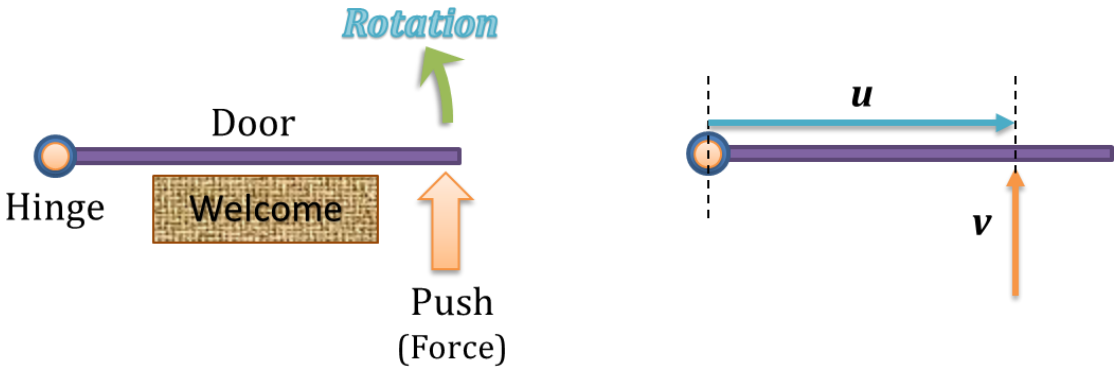
When finding work done by a force, instead of multiplying the magnitudes of two vectors together, we are multiplying the magnitude of one vector by a portion of the magnitude of the other vector. This portion is determined by the angle between the two vectors. In other words, the portion is determined by how "aligned" the two vectors are. If both vectors are in the exact same direction, or "fully aligned", then you would get the maximum output, which is similar to the product of two magnitudes. If the two vectors are not aligned at all, or in other words perpendicular (orthogonal), the output is zero.

We saw the same reasoning with both examples. In the first example, if the skills and the requirements of the region are aligned, you get the maximum output (revenue). In the second example, if the force and the displacement are aligned, you get the maximum output (work done). Any "misalignment" reduces the output.

7.8.2 Cross Product (Vector Product)

The second useful product between two vectors is called the **cross product**. As we discussed before, the only reason we define these products (or any function for that matter) is because they are useful in representing some real-world relationship. So, let's see what the cross product represents and why it is useful.

If you have ever opened a heavy door, you know that it is easier to push open if you push it closer to the handle (or knob) of the door. Instead, if you push close to the hinge, you will have to push really hard. Your push (force) opens the door and produces a rotation, with the hinge as the center of rotation, as shown below (looking from above).



We can model this situation using two vectors as shown above. Vector $\boldsymbol{v}$ represents the force you apply, and vector $\boldsymbol{u}$ represents the distance and direction from the hinge where the force is applied. As we discussed before, if you increase the magnitude of both $\boldsymbol{u}$ and $\boldsymbol{v}$, the door opens faster, producing "more rotation". The "amount of rotation a force is capable of producing" is called the "**moment of a force**", or simply the "*moment*", or **torque**. Think about it this way. If you apply a force on an object, it produces acceleration, moving in a straight line. However, if this object is anchored at a point, this force causes it to rotate around that point. The amount of rotation a force is capable of producing around a given point (center of rotation), is its "moment" or "torque". Using the door as an example, we can model the magnitude of moment as:

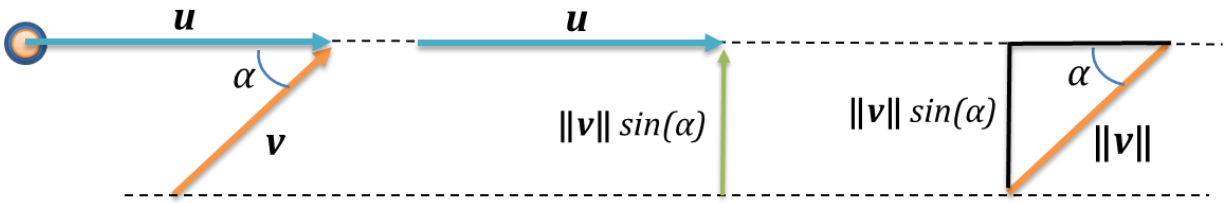
$$\text{magnitude of moment} = \text{distance from hinge} * \text{magnitude of force}$$

$$= \|\boldsymbol{u}\| * \|\boldsymbol{v}\|$$

where $\|\boldsymbol{u}\|$ and $\|\boldsymbol{v}\|$ are the magnitudes (lengths) of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, respectively.

In the above example, you place your hand perpendicular to the door to apply the force. However, what happens if you push the door with a stick at an angle, as shown on the left figure below? Again, you will find that if the stick is not perpendicular to the door, it is

harder to open the door. That is, more force is necessary to produce the same moment (rotation). Why? Because only the fraction (component) of the force that is perpendicular to the door actually contributes to the opening of the door. This fraction (component) of the force is given by the magnitude of $\boldsymbol{v}$ multiplied by sin(α), where α is the angle between the door and the force, as shown below. The middle figure below shows this component, and the right figure shows how it is calculated using a right triangle. For instance, sin(α) is 1 when α is 90^0, which means that when we apply the force perpendicularly, the entire force goes to opening the door (producing rotation). As α decreases, sin(α) decreases, thereby decreasing the fraction of force contributing to rotation. When α is 0^0, the force cannot produce any rotation because the force is parallel to the door (i.e., vector $\boldsymbol{v}$ is parallel to vector $\boldsymbol{u}$).



Therefore, we can modify the above equation for the magnitude of moment (torque) as:

$$\text{magnitude of moment} = \text{distance from hinge} * \text{magnitude of force} * \text{sine of angle}$$

$$= \|\boldsymbol{u}\| * \|\boldsymbol{v}\| * \sin(\alpha)$$

The above model summarizes what we know about opening a door. As the magnitudes of $\boldsymbol{u}$, $\boldsymbol{v}$, and sin(α) increase, we can produce more moment (and rotation). If you have ever slammed a door as a teenager, you know this very well. The same model applies to loosening or tightening a nut or bolt with a wrench. The longer the handle of the wrench ($\boldsymbol{u}$), the easier it is to turn (rotate) the nut or bolt — i.e., you have to apply less force to produce the same amount of torque. Further, it is easier to tighten a nut, if you push the wrench in a perpendicular direction to the handle of the wrench.

Are we done with our model then? Not so fast. To completely describe rotation, a magnitude alone is not sufficient. The rotation occurs about an axis. For instance, when you apply a force on a door, the door rotates around its hinges. The hinge defines a vertical axis. Therefore, the door rotates around a vertical axis. Similarly, when you apply a torque to a nut, the nut rotates about an axis. The axis is not always vertical. This axis is based on how the nut is oriented. Therefore, if you want to completely define a rotation or moment, it is not sufficient to give only a magnitude. You must specify an axis of rotation. As another example, the moment required to open a car door is very different from the moment

required to open the hood of a car or a trunk of a hatchback/minivan. Car doors have hinges with a vertical axis but hoods and hatches usually have hinges with horizontal axes.

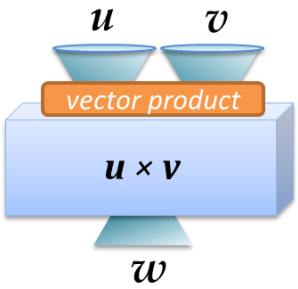
What this means is that a moment cannot be a scalar quantity. We need a direction as well to specify the axis around which rotation occurs. Therefore, we need a vector. Let's represent this vector by letter $\boldsymbol{w}$. Then the above equation becomes:

$$\|\boldsymbol{w}\| = \|\boldsymbol{u}\| \, \|\boldsymbol{v}\| \sin(\alpha)$$

Note that the above model still only gives us the magnitude of moment, $\|\boldsymbol{w}\|$. We still need to define its direction. To do that, we make the direction of $\boldsymbol{w}$ the *axis of rotation*. For instance, when opening a door, the axis of rotation is vertical. Thus, $\boldsymbol{w}$ needs to point in the vertical direction. But here is the rub: a vertical axis has two directions: up and down. We have to pick one of those directions as the "positive" direction. Since the axis of rotation is not always vertical, we cannot say that "up" is the positive direction. For instance, for the hood of a car, which side is "up"? Do you see the point? Therefore, we need a more precise definition to define the "positive" direction. To do this, we use a rule called the "right-hand" rule, which says, using your right hand, if you point your forefinger at $\boldsymbol{u}$ and middle finger at $\boldsymbol{v}$, then the positive direction of $\boldsymbol{w}$ is given by the direction of the thumb. Equivalently, if you know the direction of rotation, if you curl your four fingers of your *right* hand in the direction of rotation with your thumb sticking outwards (similar to the way you give a "thumbs-up" gesture), your thumb points in the positive direction of $\boldsymbol{w}$. For instance, when you open the door, you know the direction of rotation, so you can curl your fingers that way, leading to your thumb pointing up.

It is really important to remember that there is nothing magical about the right-hand rule. We just wanted a convention to indicate the "positive" direction of an axis of rotation. We picked one convention, which always works. This is similar to deciding that going forwards is positive and going backwards is negative. We pick a convention and stick to it, but nonetheless, it is just a convention.

Now, we have achieved what we wanted. We have found a way to calculate the torque (or moment) produced by a force $\boldsymbol{v}$, acting on a point specified by vector $\boldsymbol{u}$. As you can see, this is a vector function with two inputs, $\boldsymbol{u}$ and $\boldsymbol{v}$, producing vector output, $\boldsymbol{w}$. Since this function produces a vector output, we call it the "**vector product**". Since this is a common function, we use an operator to represent it: the × operator. That's why we also call this function the "**cross product**". Using this new operator, we can write:



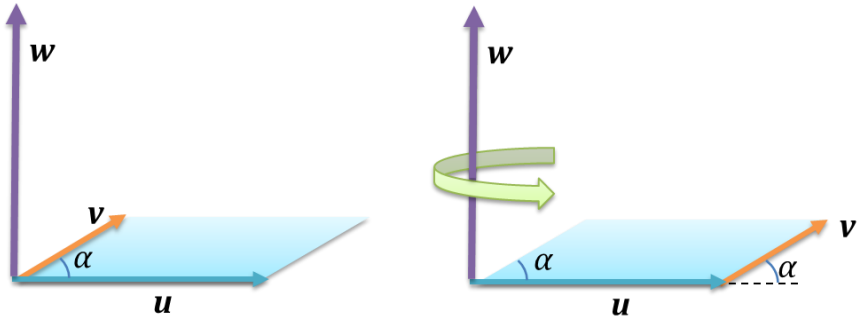
$$\boldsymbol{w} = \boldsymbol{u} \times \boldsymbol{v}$$

where the magnitude of $\boldsymbol{w}$ is given by $\|\boldsymbol{u}\|\|\boldsymbol{v}\|\sin(\alpha)$, where α is the angle between two vectors, and the direction of $\boldsymbol{w}$ is given by the right-hand rule.

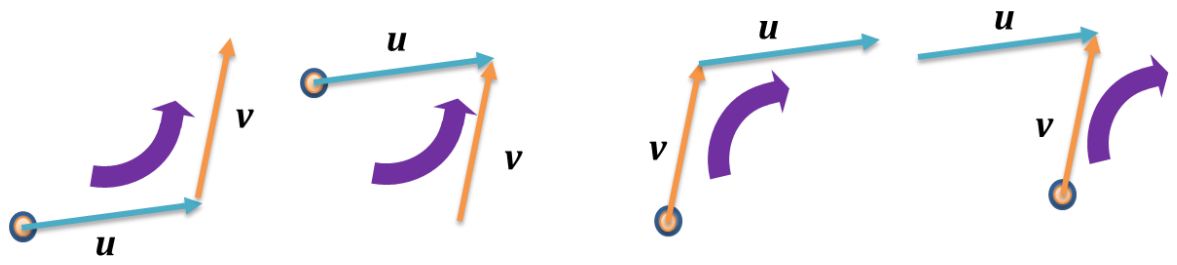
Using the definition of the cross product, let's see what increases the magnitude of moment. The magnitude of the cross product can be made larger by

(1) increasing the length of $\boldsymbol{u}$ (increasing distance from axis to the point $\boldsymbol{v}$ is applied)
(2) increasing the length of $\boldsymbol{v}$ (increasing force)
(3) making the angle between $\boldsymbol{u}$ and $\boldsymbol{v}$ close to perpendicular as possible

The following figure shows vector $\boldsymbol{w}$, resulting from the cross product between $\boldsymbol{u}$ and $\boldsymbol{v}$. In the left diagram, we have drawn $\boldsymbol{u}$ and $\boldsymbol{v}$ with their tails connected ("tails together") so that we can easily see the plane where $\boldsymbol{u}$ and $\boldsymbol{v}$ lie and the angle between them, because $\boldsymbol{u}$ and $\boldsymbol{v}$ make two sides of a parallelogram. However, note that it is much more intuitive to draw $\boldsymbol{u}$ and $\boldsymbol{v}$ "head to tail" as shown on the right because, this helps us visualize how the two vectors produce rotation. You can look at this as vector $\boldsymbol{v}$ pulling vector $\boldsymbol{u}$ to produce rotation around the axis pointed by $\boldsymbol{w}$. Just as before, vector $\boldsymbol{u}$ represents the distance from the hinge and $\boldsymbol{v}$ represents the force. However, recall that these vectors are not "grounded" at any point. Therefore, we can move them anywhere we want, and they still represent the same model. Imagining them as drawn on the right is more intuitive, although you should take special care to note the correct angle between $\boldsymbol{u}$ and $\boldsymbol{v}$.



The rotation is easy to see when you look at vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ from the top, as shown below. Figure (a) shows the two vectors drawn "head to tail", which can be thought of as vector $\boldsymbol{v}$ *pulling* vector $\boldsymbol{u}$ to produce moment (rotation). This is similar to opening a door ($\boldsymbol{u}$) by pulling it from the handle. In figure (b), the same two vectors are drawn "heads together" to show vector $\boldsymbol{v}$ *pushing* vector $\boldsymbol{u}$ to produce the same counter-clockwise rotation. This is similar to opening a door ($\boldsymbol{u}$) by pushing with vector $\boldsymbol{v}$. Since the rotation is counter-clockwise, the axis of rotation is pointing up (perpendicular to the page).



(a) (b) (c) (d)

Figures (c) and (d) show similar pulling and pushing but with vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ interchanged to show the cross product $\boldsymbol{v}\times\boldsymbol{u}$ instead of $\boldsymbol{u}\times\boldsymbol{v}$. Here, we consider the first vector as the displacement (distance & direction) and the second vector as the force. Notice that these two vectors produce clockwise rotation and you should be able to see clearly with this "head to tail" and "heads together" arrangement of vectors. Since the moments are clockwise, the result of the cross product is negative, meaning that the axis of rotation is pointing down (into the page).

7.8.3 Why is Vector Product Not Commutative?

The above figure shows an important property of the cross product: $\boldsymbol{v}\times\boldsymbol{u}$ and $\boldsymbol{u}\times\boldsymbol{v}$ do not produce the same result. Unlike scalar multiplication, where 5×2 and 2×5 give the same result, the cross product is not commutative. That is, we cannot switch the places of two inputs. Why is this the case? In the above figure, $\boldsymbol{v}$ pushing $\boldsymbol{u}$ (b) is not the same as $\boldsymbol{u}$ pushing $\boldsymbol{v}$ (d). Although both cases produce the *same magnitude* of moment (torque), they don't produce the same direction for their axes of moment (or rotation). When you switch the two vectors in a cross product, the direction of the axis changes sign. This is not the case for scalar multiplication, which represents only a magnitude. Since a cross product represents a moment about an axis, switching the two inputs changes (flips) the axis of rotation.

Now, you should have a complete, intuitive feel about what the cross product or the vector product represents.

There is another interesting observation you can make about vector representation: when we use a vector to represent a force, the direction of the vector represents the line along which the force is applied. When we use a vector to represent a torque, the direction of the vector represents the axis of rotation. This shows how the direction of a vector can be used to represent two different concepts.

7.8.4 2D Vectors vs. Complex Numbers

There is one other very interesting question you can ask about the vector product. In Chapter 6, you saw that complex numbers can represent rotated objects in a 2D plane. In fact, if you multiply one complex number by another, the product is a rotated object. How is that different from the moment of a force (torque), produced by the cross product? In other words, is the vector product similar to the product of two complex numbers? The answer is no. As we saw above, a moment produces a vector that is perpendicular to the plane where its input vectors lie. In other words, the vector product between two 2D objects cannot be represented as a 2D object in the same plane, where the inputs for the vector product come from. A complex number can never do that because the output of complex multiplication is confined to the same 2D space where the inputs come from. Therefore, it shows that, a moment of a force is not something that can be modeled using complex numbers and it shows vector functions can represent relationships that complex numbers cannot represent.

This is an important consideration in modeling 2D relationships. We have the choice to model a 2D object as a complex number or a vector. A complex number has two components similar to a 2D vector. If the functions (models) you are interested in produce outputs in the same 2D plane where the inputs come from, complex numbers are sufficient, and even preferable, because manipulating complex numbers is easier compared to manipulating vectors. For instance, we can use normal multiplication operator with complex numbers but we can't do that with vectors. That's why we heavily use complex numbers in electrical engineering where input and output waveforms can be modeled using complex numbers. However, for other types of modeling, that is not the case. That's what we saw with the moment of a force. Complex numbers are insufficient to model the moment of a force resulting from applying a force to a point away from a 'hinge'. Of course, if your objects are 3D or higher dimensioned, then vectors are the only option between the two.

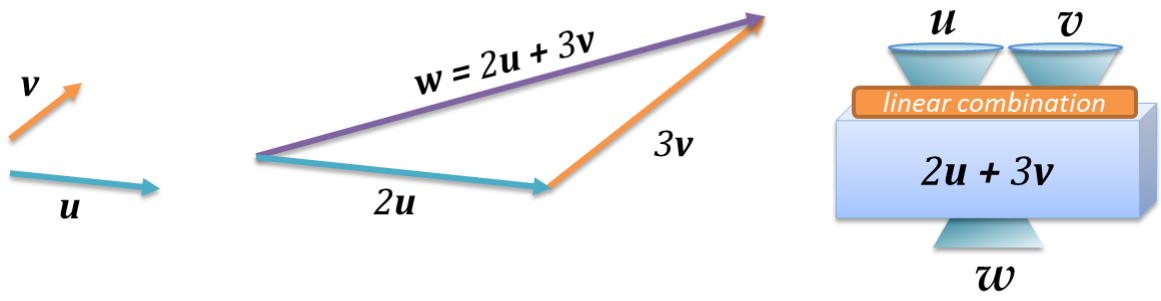
7.9 Linear Combinations of Vectors

In Section 4.2.1.2, we looked at linear combinations of **scalars**. Remember, we looked at how to make punch using different juice cans. Linear combinations with *vectors* are exactly the same, except that inputs are vectors instead of scalars. Recall that, to form a linear combination, only two operations are allowed:

(1) Inputs can be multiplied by a scalar (i.e., inputs can be scaled or amplified)
(2) The scaled inputs in (1) can be added together

Let's look at a simple example using two vectors, $\boldsymbol{u}$ and $\boldsymbol{v}$, as shown below. We form one linear combination, $2\boldsymbol{u} + 3\boldsymbol{v}$ using these two vectors, to produce the output vector $\boldsymbol{w}$. To do

that, we multiply vector $\boldsymbol{u}$ by 2, multiply vector $\boldsymbol{v}$ by 3, and add those two results together. The Visual Model for this linear combination is given on the right.



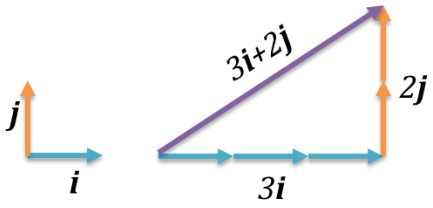
It's fairly straightforward to produce an output vector using a liner combination of input vectors. So, you may ask what the big deal is. Before I reveal the importance of it, let me ask a question: if we can change the values of coefficients 2 and 3 freely, what kind of output vectors can we produce using $\boldsymbol{u}$ and $\boldsymbol{v}$? In other words, what could $\boldsymbol{w}$ be? Think about it for a little bit. The output $\boldsymbol{w}$ can be $5\boldsymbol{u} + 3\boldsymbol{v}$, or $0.1\boldsymbol{u} + 100\boldsymbol{v}$, and so on. Is there any vector in this 2D plane that $\boldsymbol{w}$ cannot represent? The answer is no. The output vector $\boldsymbol{w}$ can become any vector you can draw on this 2D plane, as long as $\boldsymbol{u}$ and $\boldsymbol{v}$ are not parallel, which is the case for the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ shown above.

What does this really mean? Using any two non-parallel vectors ($\boldsymbol{u}$ and $\boldsymbol{v}$), we can build up (or compose) any other vector ($\boldsymbol{w}$) in the same plane, as an *output of a linear combination*. That is, vector $\boldsymbol{w}$ is a **linear combination of vectors**, $\boldsymbol{u}$ and $\boldsymbol{v}$. Conversely, any vector $\boldsymbol{w}$ in the 2D plane can be decomposed into two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ with proper scalar coefficients.

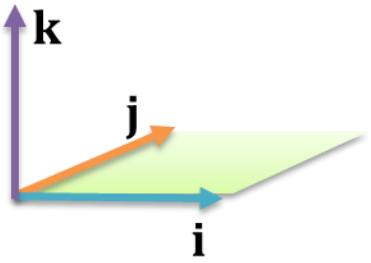
Because we can build up any other vector using $\boldsymbol{u}$ and $\boldsymbol{v}$, we call $\boldsymbol{u}$ and $\boldsymbol{v}$ "**basis vectors**" or "**basis**". As the above example shows, any two vectors can be the basis for the 2D plane the vectors reside, as long as they are not parallel. How, about 3D? We just need 3 basis vectors. How about 4D? We need just 4 basis vectors. I think you get the point.

Any vector can be represented as a linear combination of basis vectors

For the 2D plane, you can come up with any number of such basis vector pairs. One such pair is known as the **standard basis** or the **Cartesian basis**. For this basis, the two basis vectors are *perpendicular (orthogonal)*, and each have *unit length*. They are labeled **i** and **j** respectively. The following figure shows how to represent the purple vector using the standard basis vectors **i** and **j**. The purple vector can be constructed by adding 3 of **i** and 2 of **j**, to get $3\mathbf{i} + 2\mathbf{j}$.



Using basis **i** and **j**, we can write any other vector as a linear combination of these two basis vectors. For the 3rd dimension, similarly, we can introduce the basis vector **k**.



If you have been representing vectors as 5**i** + 3**j** + 2**k** without much thought, you are now well equipped to understand why we do that. Further, you should now understand that Cartesian basis is not the only basis we can have. We can pick other basis-vectors as well.

7.10 Component Representation and Algebraic Vectors

In the previous section, we introduced the basis and how we can represent any vector as a linear combination of basis vectors (or unit basis vectors in Cartesian coordinates).

Another intuitive way to think about basis vectors is as follows: if we want to specify a location on a 2D map, you can say, go 3 miles east and 2 miles north. Using "1 mile east", and "1 mile north" as our two basis vectors, we can specify any location on a 2D map. If we want to add a height, we can add another 1-unit basis vector pointing up (vertically).

This type of representation of a vector is called the **component representation** (or the component form) of a vector. In this form, we can think about a vector as a sum of its components. For instance, in 2D, the vector 2**i** + 3**j** is in the component form, and we can represent it with its two components as (2, 3) with respect to the basis **i** and **j**. In 3D, the vector 2**i** + 3**j** + 4**k** is in component form and we can represent it with its components (2, 3, 4) with respect to the basis **i**, **j**, and **k**. Since we use algebra (e.g., a sequence of numbers) to represent vectors, these vectors are also called **algebraic vectors**.

At the start of this chapter, we introduced vectors as directed line segments. We call them **geometric vectors**. We can use the component form to break down (decompose) geometric vectors into components with respect to a basis. Compared to a line segment, the component form has one huge advantage: it allows us to algebraically manipulate vectors. That's exactly what we are going to do in this section. We are going to look at the dot product and cross product using the component form.

7.10.1 Addition and Subtraction in Component Representation

Adding two vectors in the component form is straightforward. Since the basis vectors **i**, **j**, and **k** are independent (orthogonal), we can add each basis separately as shown below:

$$\begin{aligned}
\mathbf{u} &= 2\mathbf{i} + 3\mathbf{j} + 4\mathbf{k} \\
\mathbf{v} &= 5\mathbf{i} + 6\mathbf{j} + 7\mathbf{k} \\
\mathbf{u+v} &= (2+5)\mathbf{i} + (3+6)\mathbf{j} + (4+7)\mathbf{k} \\
&= 7\mathbf{i} + 9\mathbf{j} + 11\mathbf{k}
\end{aligned}$$

In other words, if we have **u** = (2, 3, 4) and **v** = (5, 6, 7), we get **u+v** = (7, 9, 11). This shows how component form leads to easy algebraic manipulations.

Vector subtraction in the component form works the same way as addition except that we perform the subtraction operation instead of addition.

7.10.2 Dot Product in Component Representation

With geometric vectors, we looked at the dot product (scalar product) between two vectors as the product of their magnitudes multiplied by the cosine of the angle between them. The component representation gives us a second definition, which is more amenable to algebra. With this definition, if we have two vectors, ***u*** and ***v***, expressed in the standard basis, the process for calculating the dot product is shown below.

$$\begin{aligned}
\mathbf{u} &= 2\mathbf{i} + 3\mathbf{j} + 4\mathbf{k} \\
\mathbf{v} &= 5\mathbf{i} + 6\mathbf{j} + 7\mathbf{k} \\
\mathbf{u}\cdot\mathbf{v} &= 2*5 + 3*6 + 4*7 \\
&= 56
\end{aligned}$$

In words, we sum up 3 terms, where each term is produced by multiplying corresponding coefficients for a given dimension. Can you explain why we do the multiplication this way? Well, in the Cartesian form, the basis vectors are perpendicular to each other. So, if you calculate the dot product of 2**i** and 5**i**, the result is 10, because the angle between the above two vectors is zero, and cos(0) is 1. However, if you multiplied 2**i** and 6**j**, the result is zero. Why? The basis vector **i** and **j** are perpendicular to each other. Therefore, $\cos(90^0)$ is 0 and

the dot product is zero. Therefore, the dot product in the standard basis boils down to a sum of products, where the products are between the same basis.

The computer code for calculating the dot product is given below for two 2D vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, where $\boldsymbol{u} = a\mathbf{i} + b\mathbf{j}$ and $\boldsymbol{v} = c\mathbf{i} + d\mathbf{j}$. The formula `a*c + b*d` calculates the dot product. The code in the 2nd box evaluates $(2\mathbf{i} + 3\mathbf{j}) \cdot (4\mathbf{i} + 5\mathbf{j})$, which evaluates to 23.

```
function dotProduct2D( a, b, c, d )
{
      sum = a*c + b*d
      return sum
}
```

```
y = dotProduct2D( 2,3,4,5 )     // (2i+3j) · (4i+5j)
print( y )                      // prints 23
```

The following function implements the dot product for two n-dimensional vectors. Consequently, it takes in two arrays, each with n elements, and n itself so that the function knows the number of elements in each array. Remember we learned how to pass arrays as function arguments in Section 5.1. In the second box, we evaluate the dot product between two 3D vectors. For 3D vectors, the length of each array is 3. The print statemen prints the result of the dot product, which is 56.

```
function dotProduct( vector1, vector2, n )
{
      sum = 0
      for i=0 to n
      {
            term = vector1[i] * vector2[i]
            sum = sum + term
      }
      return sum
}
```

```
vector1 = [2,3,4]
vector2 = [5,6,7]
y = dotProduct( vector1, vector2, 3 )   // (2,3,4).(5,6,7)
print( y )                              // prints 56
```

The computer code shows another big advantage of component representation: the dot product in the component form can be easily computed with a computer program. Since computers are used to do almost all engineering calculations today, the component

representation has a huge advantage. However, note that this component definition of the dot product does not give us any intuition as to what we are doing and why we are doing it. If you just read this section, would you be able to explain what the dot product really means? I would highly doubt it. That's why we started with geometric vectors.

7.10.2.1 Dot Product as a Linear Combination of Components

When we have two vectors a**i**+b**j** and c**i**+d**j**, we calculate the dot product as a∗c + b∗d. Notice that this formula for calculating the dot product is a *linear combination*. This is not a coincidence. In the Cartesian form, the Dot Product between two vectors is a linear combination between their scalar components. This observation has enormous implications, as we will see in the next chapter.

In Cartesian form, the Dot Product is a linear combination of components (of vectors)

There is one thing we should clarify before we move on: you can do a linear combination of different inputs — scalar inputs and vector inputs. When calculating the dot product, the inputs are **scalar** components of each vector. In such a case, the output is a scalar. We call this a **linear combination of scalar inputs**, or the dot product. Alternatively, if the inputs are vectors, like $\boldsymbol{u}$ and $\boldsymbol{v}$, you can do a linear combination of these vectors, like $2\boldsymbol{u} + 3\boldsymbol{v}$, producing an output vector, $\boldsymbol{w}$. This is a **linear combination of vector inputs**, as we saw in Section 7.9. This is *not* a dot product.

To summarize, in Cartesian form, the dot product is a linear combination of scalar components. The converse is also true. That is, when we have a linear combination with scalar inputs, that can be calculated as a dot product. We will use this fact heavily when working with matrices.

A linear combination with scalar inputs can be calculated as a Dot Product

Why is this the case? Recall that a linear combination is a linear function with more than one input. That is, a linear combination is a sum of multiple linear functions (linear terms) of single input. Consider the following linear function with one input

$$y = a\,x \qquad \textbf{(1)}$$

where x is the scalar input variable, a is the **scalar multiplier**, and y is the scalar output. Now, compare this with a linear combination of just two variables, x_1 and x_2, with two scalar constants a_1 and a_2.

$$y = a_1 x_1 + a_2 x_2 \quad \textbf{(2)}$$

$$= \boldsymbol{a} \cdot \boldsymbol{x} \quad \textbf{(3)}$$

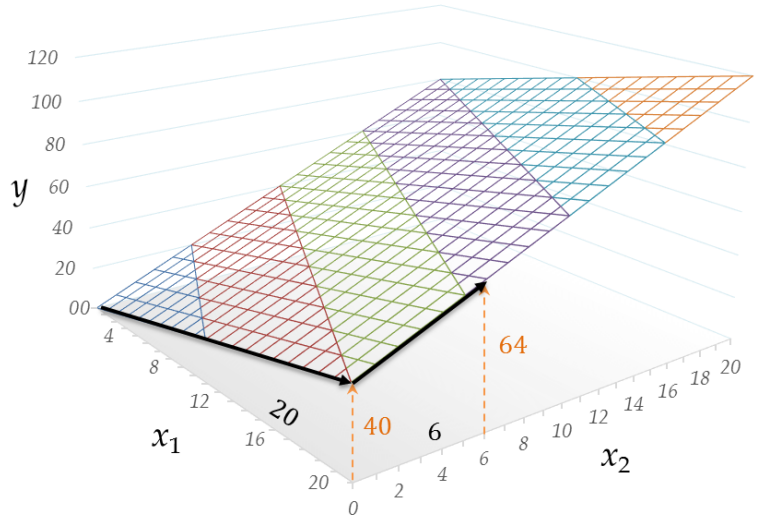
Why is (2) a dot product? If we have a constant vector $\boldsymbol{a} = (a_1, a_2)$ and input vector $\boldsymbol{x}=(x_1, x_2)$, then (2) is same as $\boldsymbol{a} \cdot \boldsymbol{x}$. In other words, vector $\boldsymbol{a}$ acts as a set of coefficients, or a **vector multiplier**, for the input vector $\boldsymbol{x}$. This shows how we can model a linear combination with scalar inputs as a dot product.

Here is another interesting observation: can you identify (1), which has just one input variable, as a degenerate case of (3), which has two input variables? Notice that (1) has a scalar multiplier and a scalar input, while (2) has a vector multiplier and a vector input. Therefore, the function $y=ax$ can also be viewed as a dot product between two vectors of just one component: (a) and (x). Therfore, the scalar multiplication, which has a single input, is a special case of the dot product. Therefore, the dot product (scalar product) is a generalization of the scalar multiplication, catering to linear combinations with multiplie scalar inputs.

The dot product in the component form is a generalization of scalar multiplication

We can also look at the dot product, and hence linear combinations with scalar inputs, graphically. You know that the graph of a linear model $y=ax$ is a straight line going through the origin, with scalar multiplier a. We can look at multiplier a as the *slope* of a line. We can extend this graph to a linear combination. How? If we were to graph the linear combination in (2), we would get a plane with a_1 as the slope in the x_1 direction and a_2 as the slope in the x_2 direction as shown below. For instance, the following graph shows

$$y = 2x_1 + 4x_2$$



The y value is the output, or the distance in y direction (height). To obtain y, we have to go in the x_1 direction along a line with slope a_1, and then go in the x_2 direction along a line with slope a_2. For instance, in the above graph, starting at the origin, we go 20 units in the x_1 direction, which has slope of 2, and then go 6 units in the x_2 direction, which has slope of 4. Therefore, the distance (height) we climbed in the y direction is $2*20 + 4*6 = 64$.

In the component form, we can look at the above dot product as follows:

total height = (slope1, slope2) · (distance1, distance2)

This expands to:

total height = slope1 ∗ disatance1 + slope2 ∗ disatance2

= height gained in x_1 direction + height gained in x_2 direction

This illustrates how a 2D linear combination is a sum of two linear terms, and similarly, how a 2D dot product is a sum of two scalar multiplications.

In this analogy, the result of the dot product is equivalent to the height we gain by climbing a linearly sloped hill described by a "slope vector" and a "distance vector". That is, the slope in each direction is described by the slope vector, and the distance we travel in each direction is described by the distance vector. Since the dot product is commutative, we can treat either the first or second vector as our slope vector. In both cases, the "other vector" becomes the distance vector. In both cases, we get the same output. Notice that we are just ascribing a meaning to inputs to get an intuitive understanding of the dot product.

This analogy teaches us two things about the dot product in the Cartesian form. First and foremost, it shows that the dot product is a linear combination of components. Conversely, a linear combination of scalar inputs can be modeled as a dot product.

7.10.3 Cross Product in Component Representation

In Section 7.8.2, we defined the cross product between two geometric vectors as the product of their magnitudes multiplied by the sine of the angle between them. Now, we are going to look at another definition of cross product, with the help of component representation. In the Cartesian form, the cross product between two 2D vectors is shown below:

$$\boldsymbol{u} = a\mathbf{i} + b\mathbf{j}$$

$$\boldsymbol{v} = c\mathbf{i} + d\mathbf{j}$$

$$\boldsymbol{u} \times \boldsymbol{v} = (ad - bc)\mathbf{k}$$

This can be derived algebraically as follows:

$$\boldsymbol{u} \times \boldsymbol{v} = (a\mathbf{i} + b\mathbf{j}) \times (c\mathbf{i} + d\mathbf{j})$$

$$= a\mathbf{i}\times(c\mathbf{i} + d\mathbf{j}) + b\mathbf{j}\times(c\mathbf{i} + d\mathbf{j}) \quad \textbf{(1)}$$

$$= ac\,(\mathbf{i} \times \mathbf{i}) + ad\,(\mathbf{i} \times \mathbf{j}) + bc\,(\mathbf{j} \times \mathbf{i}) + bd(\mathbf{j} \times \mathbf{j}) \quad \textbf{(2)}$$

Note that $\mathbf{i} \times \mathbf{i}$ and $\mathbf{j} \times \mathbf{j}$ are zero, because when two vectors are parallel, their cross product (torque) is zero, because the angle between them is zero (therefore the sine of the angle is zero). Therefore, (2) becomes:

$$\boldsymbol{u} \times \boldsymbol{v} = ad(\mathbf{i} \times \mathbf{j}) + bc(\mathbf{j} \times \mathbf{i}) \quad \textbf{(3)}$$

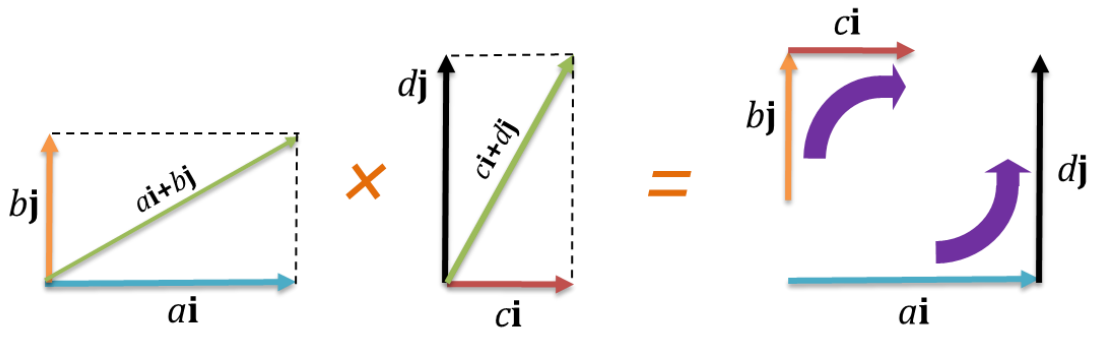
Further, if you look at $\mathbf{i} \times \mathbf{j}$, the result is $\mathbf{k}$. Remember, the cross product produces a vector that is perpendicular to both input vectors. What is perpendicular to both $\mathbf{i}$ and $\mathbf{j}$? It is $\mathbf{k}$. Alternatively, if you use the right-hand rule, with your index and middle finger pointing in the direction of $\mathbf{i}$ and $\mathbf{j}$ respectively, then your thumb points in the direction of $\mathbf{k}$. Similarly, $\mathbf{j} \times \mathbf{i}$ is $-\mathbf{k}$, using the same right-hand rule. Therefore, from (3), we get:

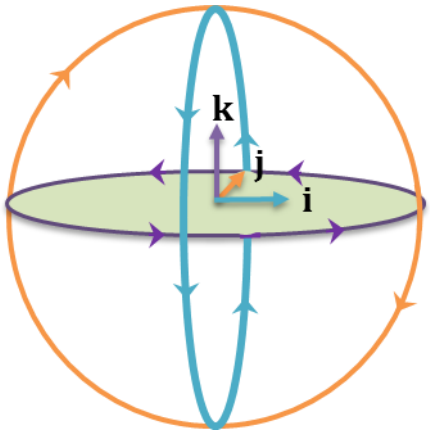
$$\boldsymbol{u} \times \boldsymbol{v} = (ad - bc)\mathbf{k}$$

What does this new vector represent? In order to understand that, let's draw the two vectors $a\mathbf{i}+b\mathbf{j}$ and $c\mathbf{i}+d\mathbf{j}$ as given below. From (1), ignoring the zero terms, we get the cross product:

$$\boldsymbol{u} \times \boldsymbol{v} = a\mathbf{i} \times d\mathbf{j} + b\mathbf{j} \times c\mathbf{i}$$

This multiplication is illustrated in the figure below. Remember, we looked at two vectors producing this type of rotation in Section 7.8.2. You can see that the term a**i** × d**j** produces a moment (rotation) in the counter-clockwise direction (looking from top). Its magnitude is *ad*, and its axis of rotation is **k**, from the right-hand rule. The other term, b**j** × c**i** produces rotation in the clockwise direction. That is, its axis of rotation is -**k**. The net moment is determined by the difference of these two terms. If the magnitude of *ad* is larger than the magnitude of *bc*, the counter-clockwise moment wins, and these two vectors produce a counter-clockwise moment (rotation). If the magnitude of *bc* is larger than the magnitude of *ad*, then we will have a clockwise moment (rotation).





If the vectors are 3 dimensional, you can use similar algebraic expansion and reasoning to understand the cross product. The only difference is that now you have rotation around all 3 axes. Can you imagine a sphere rotating around all 3 axes? It is shown here. When you hit a baseball with a bat, or use a cue stick to hit the cue ball in pool (billiards), it is likely that the ball will spin around all 3 axes. So, the next time you hit a baseball with a bat, remember that you are taking the cross product between the force delivered by the bat and the vector representing the point you hit on the ball (from its center of gravity). However, don't tell anyone on the ball field that you are calculating cross products, if you have any desire to play baseball again.

7.11 What Unites Algebraic and Geometric Vectors?

We saw that we can model geometric objects in 3D space using vectors. However, do vectors always have to be geometric objects in Euclidian space? Let's see. Let's say you want to represent your SAT score, which has two components — a math score and a verbal score. You cannot use a single number to represent such a score with two components.

Let's take another example. In team sports like soccer, a score is expressed as 5 to 3 (team A has 5 points and team B has 3 points). This score has two components. Similarly, we can represent a pant by numbers 30 and 36, where the 1st number is the waist and 2nd number is the length. To indicate that these two numbers are related and belong to the same object, we use parenthesis to group them as (30, 36) to create one vector. This is an **algebraic vector**, as we saw in Section 7.10. Notice that vector (30, 36) has nothing do with geometry. It does **not** represent coordinates on a plane. It's just a pair of numbers. Of course, we can plot a pair of numbers as coordinates on a plane, but that's just a visualization aid. You don't need to plot the waist and length of a pant to describe a pant.

An Algebraic Vector is just a sequence of numbers

An algebraic vector with 2 components (2 real numbers) is said to be in $\mathbb{R}^2$ space (analogues to 2D for geometric vectors). Similarly, we have $\mathbb{R}^3$ (for 3 numbers), and, by extension, $\mathbb{R}^n$ (for n-numbers). For instance, the group of three numbers (100, 113, 112) is a vector in $\mathbb{R}^3$, which could describe Red, Green, and Blue (RGB) primary color values

A sequence of n numbers represents a vector in $\mathbb{R}^n$

(components) of a compound color.

It is important to understand the difference between geometric vectors (line segments in Euclidian space) and algebraic vectors (a sequence of numbers). Then, why are they both called vectors? It's because both of them *can be composed with linear combinations*.

A vector is a linear combination (of some basis)

Vectors are nothing more than linear combinations, of some basis. Equivalently, vectors can be linearly combined to produce other vectors. Therefore, we can expand the definition of vectors to include any object that can be expressed as a linear combination. Both geometric vectors and algebraic vectors fit the bill because both of those can be composed using components (i.e., basis). A line segment in Euclidian space can be constructed by linearly combining different line segments (see Section 7.9). Similarly, for algebraic vectors, take the RGB vector (100, 113, 112) we saw before. It can be modelled as a linear combination as:

$$(100, 113, 112) = 100*(1, 0, 0) + 113*(0, 1, 0) + 112*(0, 0, 1)$$

Here, (1, 0, 0), (0, 1, 0), and (0, 0, 1) are the basis vectors, each representing Red, Green, and Blue.

Since geometric and algebraic vectors are both linear combinations, we can easily convert between them. For instance, if we have an algebraic vector (3, 5), we can plot it on a 2D plane as a directed line segment. Similarly, when we have a directed line segment, if we decompose it to components with respect to some basis (e.g., as 2**i** + 3**j** in the standard basis), we can represent that as an algebraic vector (2, 3).

This can be a major source of confusion for many students. As someone quipped, on Monday, a vector is a directed line segment; on Wednesday, a vector is a sequence of numbers. Now you know better — a vector is a linear combination of some basis. For instance, a punch we make using 3, 5, and 6 cans of apple, mango, and orange juice, respectively, can be represented as (3, 5, 6), with respect to the above 3 varieties of juice cans as the basis.

If you understood that, here is a brain teaser. We learned that a polynomial is a linear combination of power functions. Are polynomials vectors as well? You bet they are. What is the basis of those vectors? Yes, power functions. For instance, the vector (3, 5, 2) represents the polynomial $3x^2 + 5x + 2$, with respect to the basis (power functions) x^2, x^1, and x^0. If you wanted to add $8x^2 + 5$ to the above polynomial, you could add vectors (3, 5, 2) and (8, 0, 5) together. Similar reasoning easily shows complex numbers as vectors with a real part and an imaginary part as the basis.

Polynomials and Complex Numbers are Vectors too!

7.12 Modeling with Vectors (Scalar Fields and Vector Fields)

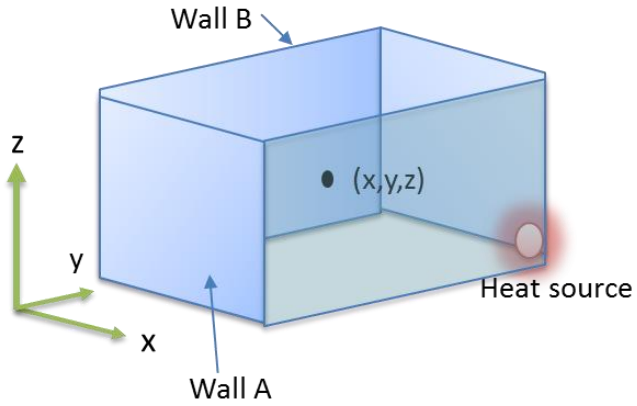
Vectors allow us to model relationships with multiple inputs and multiple outputs. Let's see how we can do that.

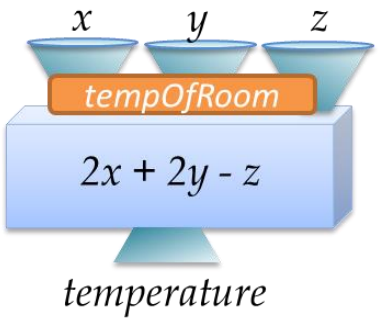
7.12.1 Scalar Fields

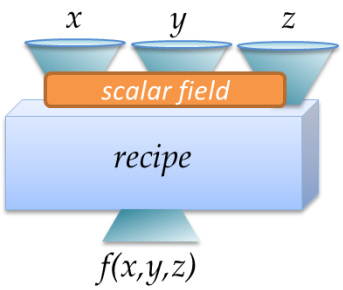
As we saw in Section 1.8, real-world models can have many inputs. However, there is an alternate way to represent multiple inputs. Can you guess what it is? Yes, a vector. A vector can contain multiple scalar components. Therefore, a function accepting multiple inputs can be modelled as a function accepting a vector.

Let's look at an example. We can represent the temperature at each point in a room with a function of 3 variables as follows:

$$\text{tempOfRoom}(x, y, z) = 2x + 2y - z$$







In the above function, the inputs x, y, z represent each point (x, y, z) in 3D space, with reference to the Cartesian coordinate system shown. For instance, x represents distance from wall B, y represents the distance from wall A, and z represents the height from the floor. According to the function definition, as we get away from walls A and B (as x and y increase), the temperature rises. Similarly, as we go up in height (as z increases), the temperature drops. This could mean that there is some heat source (like a fireplace) on the floor, at the opposite corner from where wall A and B meets (as shown). Do you see how we can use functions of multiple inputs for modelling?

The output of this model, temperature, is a **scalar** quantity. Therefore, the function tempOfRoom(x, y, z), which outputs a scalar value (e.g., temperature) for each input point in 3D space, is called a "**scalar field**". The term "field" refers to the 3D space. Therefore, a "scalar field" represents a 3D space that is filled with scalars, which happen to be temperature values in this example.

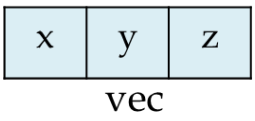
The computer code for function tempOfRoom is given below, along with a call to this function to evaluate the temperature in the room at the point (5, 3, 6).

```
function tempOfRoom( x, y, z )
{
      temp = 2*x + 2*y - z
      return temp
}
```

```
temp = tempOfRoom( 5,3, 6 ) // temp at point 5,3,6
print( temp )               // prints 22
```

The function tempOfRoom(x, y, z) accepts 3 scalar inputs. However, the point (x, y, z) can also be thought of a vector with 3 components. This is a **position vector**. The Visual Model shows a model accepting a position vector, $\boldsymbol{u}$, and producing a scalar value, s. To model our temperature field, position vector $\boldsymbol{u}$ needs three scalar components: x, y, and z.

The computer code below shows the `tempOfRoom` function implemented using a single input "vector", which is represented by an *object* called "`vec`". In programming, an object is a collection of multiple data items. Our object `vec` has 3 components to represent the x, y, and z components of the vector. Notice that we use the "." (dot) to access each component (member) of the object. For instance, `vec.x` refers to the `x` component of the `vec` object. When we call the function, we use the JavaScript-like expression `{x:5, y:3, z:6}` to set each component of the `pos_vec` object. Note that different programming languages have different constructs like structures, classes, and objects to group multiple components into one entity. We could have also used a 3-element array to represent our 3D position vector but objects/structures give us the ability to name each component like `x`, `y`, and `z`.



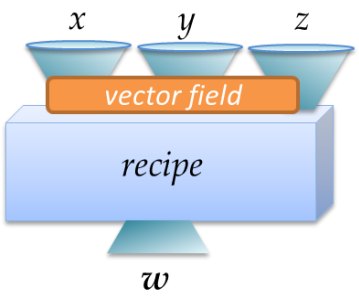
Most importantly, the code above treats the input to the function `tempOfRoom` as 3 scalars, while the code below treats the input as a single vector. This clearly shows both ways we can represent the input to a scalar field.

```
function tempOfRoom( vec )    // one input with 3 components
{
      temp = 2 * vec.x + 2 * vec.y - vec.z
      return temp
}
```

```
pos_vec = {x:5, y:3, z:6}    // create position vector (5,3,6)
temp = tempOfRoom( pos_vec ) // find temp at position vector
print( temp )                // prints 22
```

7.12.2 Vector Fields

In the previous section, we met scalar fields. However, as we know, a scalar cannot represent all objects we want to represent. For instance, what if we wanted to represent the velocity of wind at each point in a room? Velocity is a vector. This means that each point in space is assigned a vector. In other words, each input (x, y, z) point has a corresponding output vector value, $\boldsymbol{w}$, as shown by our Visual Model. Such a field is called a "**vector field**".



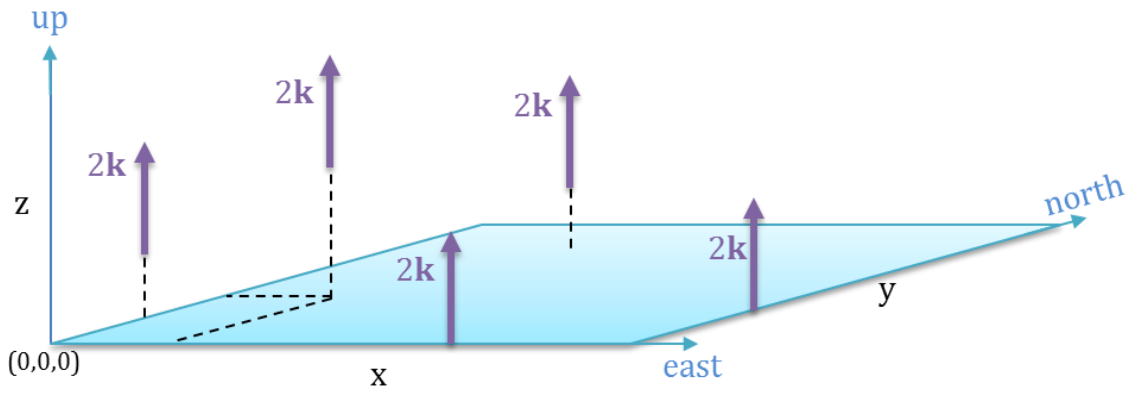
Vector fields are a vast area of study. I just wanted to introduce you to the concept and show how it fits in the bigger picture. You may wonder what the point is in knowing the velocity of air at every point in a room. Well, that's effectively how you would model a wind tunnel, or a tornado, which is modelled at a much large scale than a room. The same concept applies to air or other fluids flowing through pipes, aircraft engines, turbines, etc. Velocity is not the only vector we want to associate with points in space. Another quite useful vector that can be associated with a point in space is force. Such a vector field is also known as a "**force field**". The force can be due to electrical charge, magnetism, gravity, etc. If you hear people referring to a gravitational field or an electric field, now you know what they mean.

One last example. Say you are a hiker climbing a steep slope on a trail, and you stop at some point on the trail. If you go east, you have to climb a 10^0 slope. If you go north, you have to climb a 5^0 slope. If you go west, you have to climb down 9^0 slope, and so on. As you can see, the slope changes based on the direction. Therefore, to represent such a slope, we need a vector. This vector is known as the **gradient**. In component form, the gradient gives the slope in the direction of each component. As a geometric vector, it points in the direction of the maximum slope at that point. Therefore, to represent the gradient at *each point*, we need a vector field, which is also known in this case as a **gradient field**.

To develop an intuitive understanding about how to model a vector field, let's consider an example with velocity of air over an open flat surface, like a tennis court or a parking lot. Let's start small. First, let's say that air is rising up uniformly over the parking lot, because the surface is uniformly heated. This is shown below.

If we model the above situation with a vector filed, we would have something like,

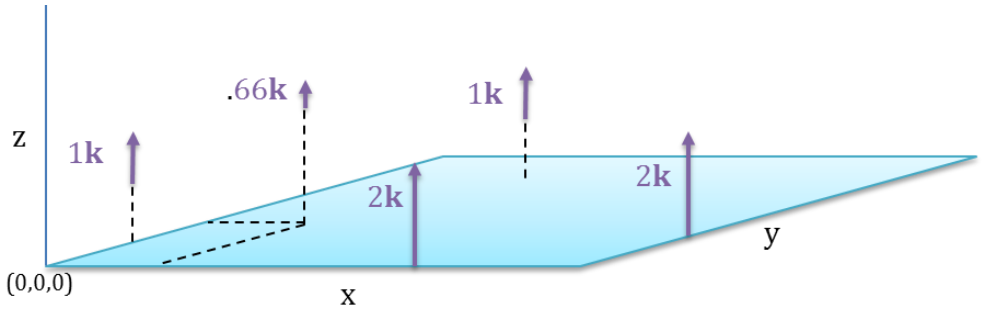
$$\text{velocityOfAir}(x, y, z) = 0\mathbf{i} + 0\mathbf{j} + 2\mathbf{k}$$



The above model shows that air is moving straight up at each point in space, at a constant velocity of 2 miles an hour. The figure above shows the velocity for some random points. A point can be thought of as x units east, y units north, and z units up, measured from the south-west corner of the parking lot, which is treated as the origin, (0, 0, 0). At any point (x, y, z), air is rising vertically at a constant velocity. The vector $2\mathbf{k}$, at each point, represents this fact. Note that the air does not move horizontally at all.

Now, let's make our model a little bit more interesting. Let's say that air closer to the surface rises faster. To model this, let's say that the velocity of air is 2 miles per hour at the surface and is inversely proportional to the height from the surface as given below:

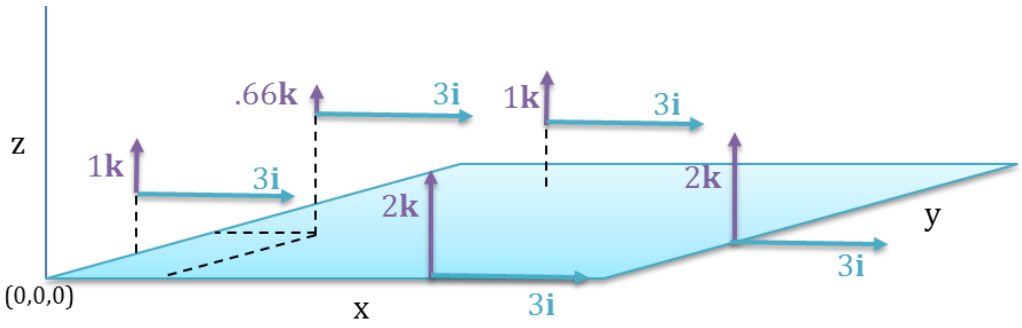
$$\text{velocityOfAir}(x, y, z) = 0\mathbf{i} + 0\mathbf{j} + 2/(z+1)\mathbf{k}$$



Velocity in the direction of **k** is now given by the function $2/(z+1)$. For instance, when the z coordinate is 2, we get $2/(2+1) = 2/3$. Note that the velocity is lower as we go higher. However, the air is still rising only vertically.

Now, let's make this model even a little bit more interesting. Let's add some wind blowing from west to east. Say this wind has a constant velocity of 3 miles per hour. Now, we get:

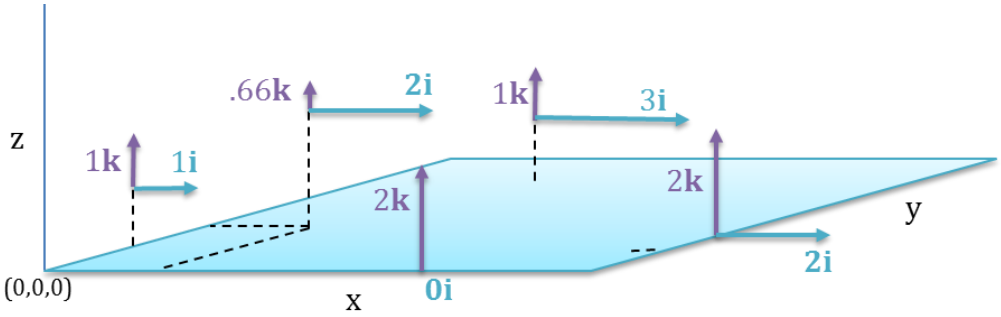
$$\text{velocityOfAir}(x, y, z) = 3\mathbf{i} + 0\mathbf{j} + 2/(z+1)\mathbf{k}$$



In the above figure, the velocity at each point has a horizontal component. This means that air is moving east as well as upwards. In fact, this eastward velocity is higher than the upwards velocity at every point, so air is going eastwards faster compared to rising up.

If the wind from the west was not constant, but instead increased as we went north, linearly, our model would become:

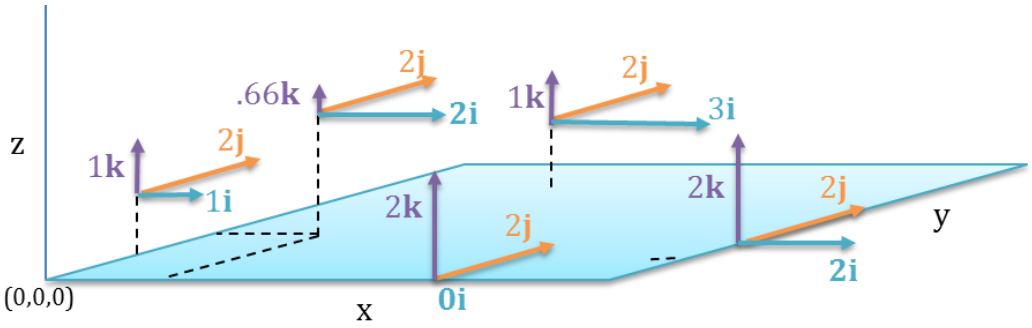
$$\text{velocityOfAir}(x, y, z) = y\mathbf{i} + 0\mathbf{j} + 2/(z+1)\mathbf{k}$$



Note that there is no wind blowing eastwards at the south edge of the parking lot. However, as you walk north, you will experience increasing eastward wind. Notice how the **i** component grows as you go north (as we increase y).

Now, let's add a constant cross-wind to the above mix, blowing from the south to north. Then our model becomes:

velocityOfAir(x, y, z) = y**i** + 2**j** + 2/(z+1)**k**



As you can see, at every point (except the points at the south edge), air is moving north, east, and up at the same time. The exact direction of wind at each point is determined by the corresponding component of the vector at that point.

The velocity at any point can be calculated by plugging in the coordinates of that point — i.e., using coordinates as input values to the function. For instance, the velocity at point (1, 2, 2) is given by:

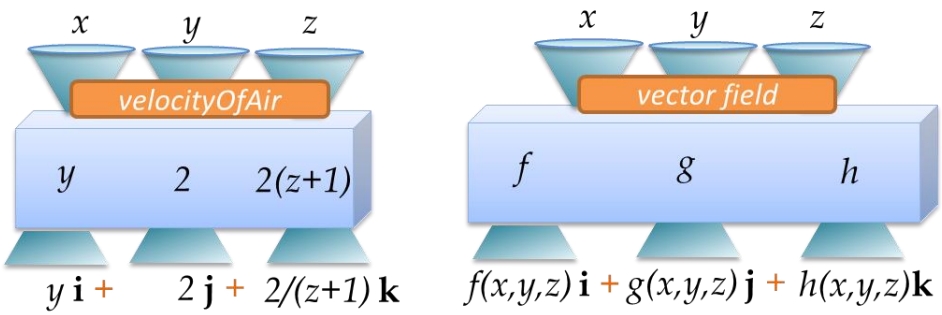
velocityOfAir(1, 2, 2) = 2**i** + 2**j** + 2/(2+1)**k** **(1)**

= 2**i** + 2**j** + (2/3)**k**

This point (1, 2, 2) is also shown in the above figure, at the end of the dashed lines.

This process shows how you come up with a vector field. Although you may think that it is silly to model how air moves over a parking lot, similar vector fields help scientists model weather over a region. The movement of air is important for predicting weather, the path of a tornado, or a storm. That's why we learn about these models.

There is another point I want to make about this vector filed. Look at the coefficient of each basis vector in (1). This coefficient is itself a scalar function. For example, the coefficient of basis **i** is y, which is a scalar function of y. The coefficient of basis **k** is $2/(z+1)$, which is a scalar function of z. In general, the coefficient can be a function of x, y, and z. Since each coefficient can be a scalar function, we need 3 scalar functions to define a vector field in 3D space. The Visual Models below shows this view. The Visual Model on the left shows our VelocityOfAir vector field while the one on the right shows a general vector field of 3 input variables.

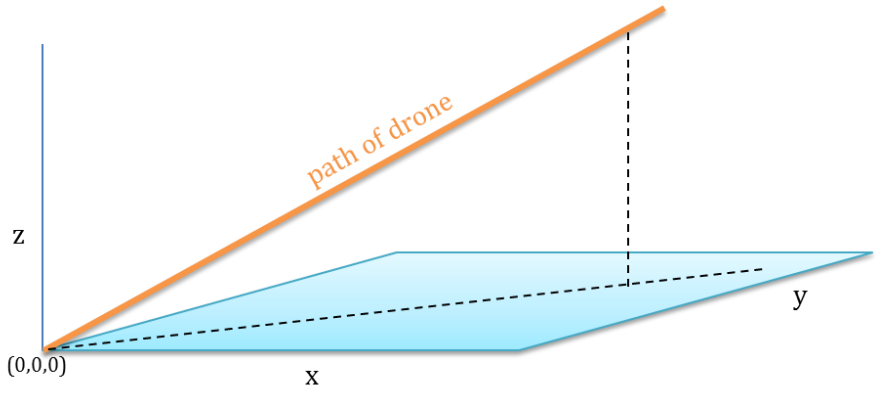


We can look at the 3 input coordinates, (x, y, z) as an input vector as well. You can think of this as a position vector, representing each position in the input space. The main reason I explained this view is because of the next chapter describing matrices. Using matrices, we can accept input vectors with multiple components and produce outputs with multiple components. The above Visual Models describe why we need this capability — because, vector functions need to accept vector inputs with multiple components and produce output vectors with multiple components.

7.12.3 Vector Valued Functions of a Parameter

Any function that outputs a vector is a **vector valued function**. A vector field, which is a vector valued function, has multiple inputs. However, what if the inputs are related, as we saw in Section 1.11? Then, we can use a common parameter to represent all inputs. For instance, in our previous example, if all of our inputs are on a straight line, or on a surface, the inputs are related and we can use a common parameter. For instance, let's say we want to send a drone over our parking lot along the main diagonal of the parking lot, starting from the origin. This diagonal is a straight line, as shown below, where $x = y = z$. For this,

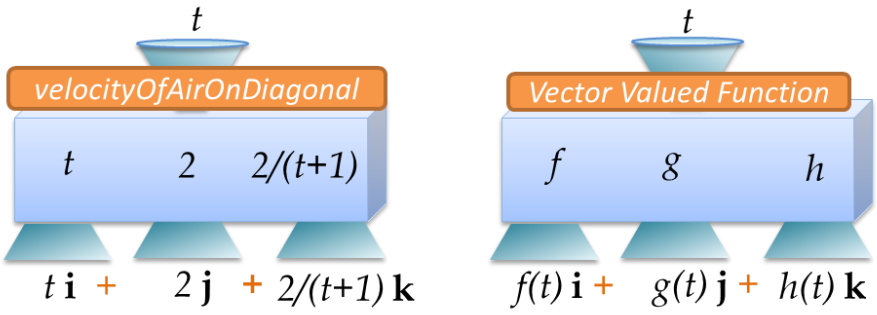
we are only interested in the velocity of air along this diagonal. We don't have to know the velocity at every point in entire 3D space over the parking lot. That's overkill.



We can use the parameter t, to represent this path, by setting $t = x = y = z$. The Visual Model below, on the left, models this vector valued function. Notice that we replace x, y, and z with t to obtain this model, and hence, it is a model with just one input variable, t.

A function with a vector output is a Vector Valued Function

The Visual Model for a general vector valued function of one input is given on the right. Notice that each component is now obtained with a scalar function of t. As you can see, this is a much simpler model than the general vector field, because it has only one input variable.



7.13 Function Space

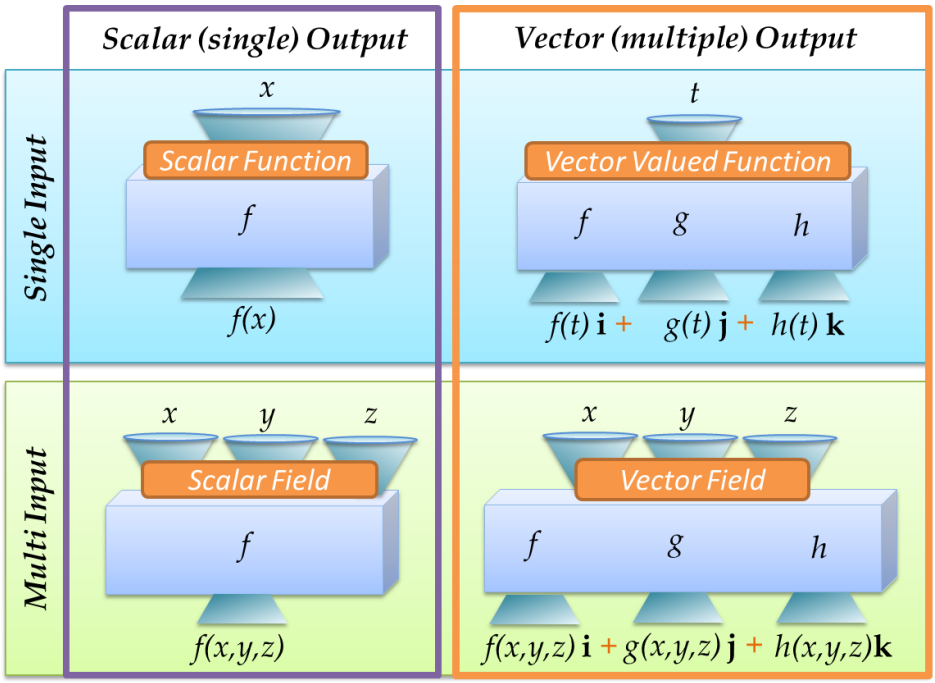
In this book, we have looked functions that accept only one input and functions that accept multiple inputs (vectors). Similarly, we have looked at functions that produce single

(scalar) output or outputs with multiple components (vectors). Therefore, we can categorize all of these functions into four categories, as shown below. Notice that functions that accept multiple inputs (a point in space) are usually called fields.

	Single Output ($\mathbb{R}$) (Scalar Output)	**Multi-Component Output ($\mathbb{R}^n$) (Vector Output)**
Single Input ($\mathbb{R}$)	Scalar Valued Function (or Scalar Function)	Vector Valued Function
Multi-component (Vector) Input ($\mathbb{R}^n$)	Scalar Field	Vector Field

Remember that we represent a single input with $\mathbb{R}$ and n inputs with $\mathbb{R}^n$. The letter $\mathbb{R}$ represents the set of real numbers. A function that accepts one scalar input accepts any number in $\mathbb{R}$ (any real number). A function that accepts two inputs accepts an input from the set of real numbers $\mathbb{R}\times\mathbb{R}$ or $\mathbb{R}^2$, because the input set is a set of vectors like (1, 1), (1, 2), (2, 1), etc., which describes $\mathbb{R}^2$ combinations (tuples). Similarly, a function accepting n inputs is represented by $\mathbb{R}^n$. The same goes for the output. Note that an n-dimensional vector is equivalent to $\mathbb{R}^n$ in its component representation, because it has n scalar components.

The following figure shows the same categorization we did above, but with Visual Models. In these, each letter f, g, or h, represents a scalar function.



The above figure nicely summarizes which type of model we should use to model a given situation. That is, we can now easily pick a model based on the type of input we have and the type of output we want to produce.

7.14 The Story So Far

In the previous chapters, we got a glimpse at the function dynasty: how functions came into existence, their common properties, their common problems,, and the objects they consume and produce. We also looked at the most common models, and how an infinite series of simpler functions can model more sophisticated functions. In the previous chapter, we extended the objects that functions consume and produce to include "rotated" objects in 2D, or objects with two orthogonal components.

In this chapter, we again extended the objects that functions consume and produce. Instead of objects with just a magnitude, or rotated objects in the 2D plane, we met vectors, which could represent objects in any orientation in one, two, three, or higher dimensional space. In 3D space, they appear as directed line segments. Alternatively, we saw that we can represent any such vector with a sequence of 3 numbers as well. Most importantly, we saw that vectors are just linear combinations of some basis. According to this definition, line segments, sequences of numbers, polynomials, and complex numbers are all vectors composed of different basis.

As an example, if a scalar could represent your proficiency in one skill, a vector could model your proficiency in many skills. Instead of representing you with just one skill, vectors would allow us to represent many of your skills, and hence is a better representation of you. A function accepting such a vector as input can predict how well you can do in a given career, based on the skills that career requires. Therefore, with these new objects called vectors, there came new functions too — the dot product (scalar product) and the cross product (vector product). We looked at what each of them could model. We saw that the dot product calculates the "combined strength" of two vectors, whereas the cross product calculated the "tendency to produce rotation" when two vectors are considered.

Finally, we looked at how to create models with vectors. First, we saw how we can have a vector as an input to a model. Such a model can produce a scalar output, yielding a scalar field, or produce a vector output, yielding a vector field. With scalar fields, we can model the distribution of a scalar quantity like temperature in space. If a model produces a vector output instead, for each input point in 3D space to represent a vector output like velocity, acceleration, gradient, etc., we get a vector field. In component representation, such a vector field looks like a group of multiple scalar functions, where each function is calculating one output component using multiple inputs. In essence, functions producing vector output use a group of scalar functions to act as a combined function, where each scalar function produces one component of the output vector using the same inputs.

If you have ever listened to film or literary critics, you've probably heard expressions like "that character in novel X lacks depth", "it is a single dimensional character", or a "two-dimensional character". Another character may be described as "multi-dimensional", showing a lot of nuances and complicated behavior. This is exactly the difference between scalars and vectors. Vectors let us capture many dimensions. If a novel has a "flat" character, the behavior the character exhibits will be "flat" as well. A villain will always kill. The protagonist will always do the right thing. A character filled with many complicated attributes, on the other hand, will react to situations in quite subtle and nuanced ways. That's exactly what vectors and vector functions allow us to model.

8 MATRICES: EXTENDING LINEAR FUNCTIONS

Chapter Overview: Linear models are the easiest models for us to build and analyze and this chapter looks at how to extend linear models so that they can accept vector inputs and produce vector outputs. In the process, we will meet our main protagonist, the matrix, and its ability to perform linear transformations.

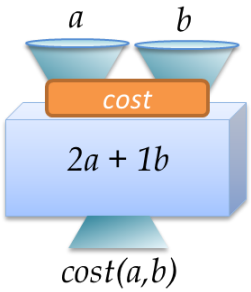
In this chapter, we are going to take linear combinations to a new level. The first part of this chapter is written in a way so that you can understand it even if you don't have a deep knowledge of vectors. This will allow everyone to appreciate the importance of vectors.

8.1 Linear Combination Revisited

You use linear combinations almost every day. If an apple is \$2 and a banana is \$1, the cost of 5 apples and 4 bananas is given by $5*2 + 4*1$. More generally, the cost of a apples and b bananas is given by:

$$2a + 1b \qquad \textbf{(1)}$$

This is a **linear combination**. That is, we multiply each input variable (a and b) by a coefficient and add the results together. This linear combination can be used to find the cost of any number of apples and bananas. Therefore, we can represent it as a *function* of two input variables:



$$y = 2a + b, \quad \text{or}$$

$$\text{cost}(a, b) = 2a + b \quad \text{[a linear combination is a function]}$$

With a linear combination, we can pick some fraction of each input and add them together. You can think of this as making apple juice from concentrate. You can mix x parts of water (say, 4 cups) with y parts (say, 1 cup) of apple juice concentrate to make apple juice. Now, your 5 cups of apple juice are actually a linear combination of water and concentrate. Every time you drink apple juice, you are drinking a linear combination!

We can also define linear combinations of more variables. E.g., if a price of a cantaloupe is \$3, for c number of cantaloupes, we can expand our linear combination as:

$$y = f(a, b, c) = 2a + b + 3c$$

Similarly, we can also use a linear combination of water, apple juice concentrate, and orange juice concentrate, to make punch.

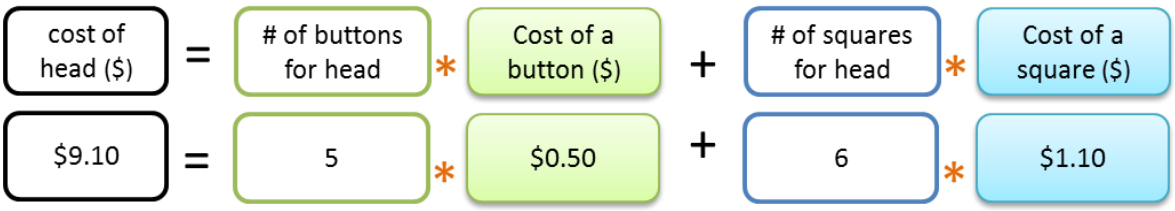
Quick quiz: what's a linear combination of just one input variable? Well, it's a linear function of the form $f(x) = ax$. Therefore, you can look at a linear combination as a sum of multiple linear functions. That is, the price of apples is given by one linear function, and the price of bananas is given by another linear function. When you add these two functions together, you get the linear combination in (1). It is important to understand that a linear combination is a generalization of the linear function $f(x) = ax$ to multiple variables.

A linear combination is a generalization of the linear function $f(x) = ax$

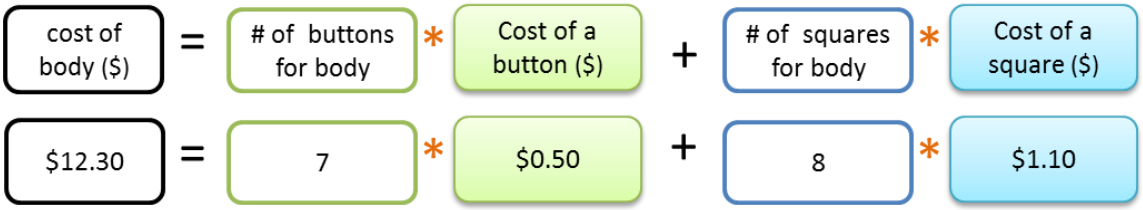
8.2 Multiple Linear Combinations

Let's say we start a business where we make teddy bears, as shown below. Each teddy bear has a head and a body. The head is made with cloth and buttons. So is the body — not very attractive, eh, but work with me here. I am trying to explain math; not how to make hand craft to sell online.

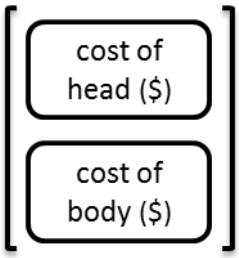
Say that the cost of a button is $0.50, and a square piece of cloth is $1.10. We need 5 buttons and 6 squares of cloth for the head. Therefore, the combined cost to create head is (in $):



Similarly, if we need 7 buttons and 8 squares of cloth for the body, the combined cost to create the body, (in $):



Now, we have two linear combinations. Not just one! The cost to create the teddy bear is the "cost of head" plus "cost of body" but we want to track these two costs separately, because we may change how the head or the body is made — e.g., we may switch to five buttons for the body instead of seven.

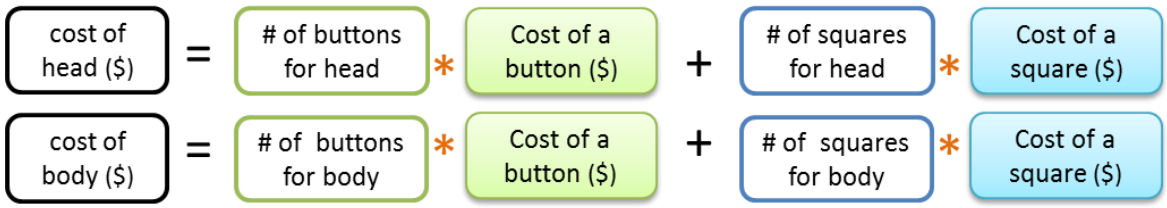


Thus, we track these two costs as a sequence of two numbers (called a **vector** in $\mathbb{R}^2$, because it has two scalar components). Remember, we looked at vectors, specifically *algebraic* vectors in $\mathbb{R}^n$, in the previous chapter. Remember, we use numbers to represent objects. In this case, we need more than one number to represent the cost of a teddy bear, if we want to track the cost of the head and the body as two separate costs. At the same time, the two numbers are related. They belong to the same teddy bear. Therefore, we want to keep the two numbers (costs) grouped as a single unit. This group of numbers (in a given order) is a vector, as shown to the right, as a single column. In our case, since it represents two different costs, we call it the "**cost vector**". Each cost (for head or body) is called a *component* of that cost vector.

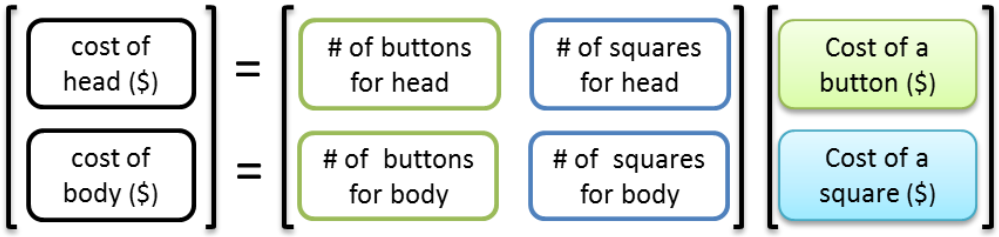
A Group of Numbers (in a specific order) is a Vector

To come up with our cost vector, we grouped the outputs of two linear combinations (functions) for the head and the body together. Remember, we need two functions to produce two outputs, because a function can produce only one output. However, these two functions (linear combinations) are closely related because they represent how to obtain

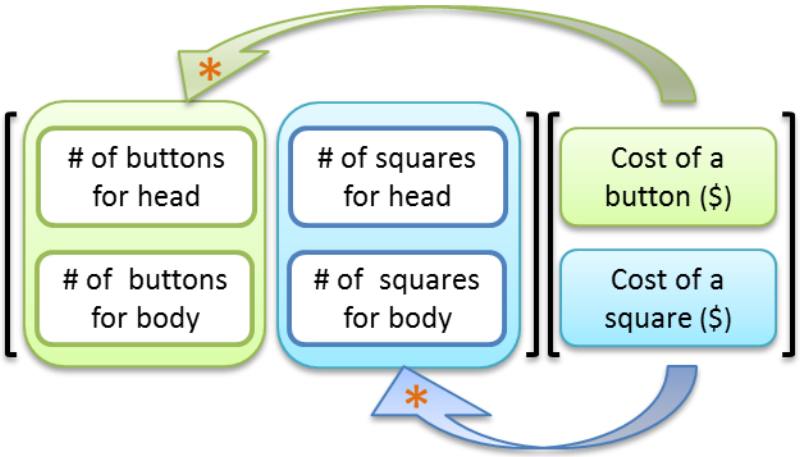
the cost of one teddy bear (i.e., a recipe for producing the cost of a teddy bear). Moreover, these two functions have the same two inputs — the cost of a button and the cost of a square (piece of cloth). So, can we keep these two functions together as a single relationship (or a single recipe)? To answer that question, let's take another look at our two linear combinations (functions) together:



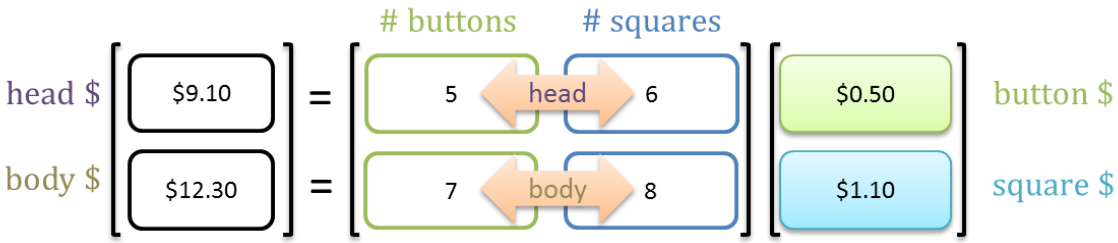
If you look at the above two linear combinations carefully, you should see that the "cost of a button" (green-filled box) and the "cost of a square" (blue-filled box) appear in both equations. In simple algebra, what do we do when we have a common term appearing multiple times? You guessed right: we factor it out. We can do the same thing here, using the following notation.



The above is called the "matrix form" of the two linear combinations, because we have this new object with 4 components, called a "**matrix**". Before we find out exactly what a matrix is, pay careful attention to how we got this matrix form from our two linear combinations. Given a matrix form like this, understand what needs to be done to get the original two linear combinations. It is very important to understand this point: this matrix form is another way to write our two linear combinations, and hence, we should be able to go back and forth from one from to the other easily. The following figure shows exactly how that is done. The first column shown in green-outlined boxes (i.e., number of buttons), needs to be multiplied by the green-filled "cost of a button". Similarly, the blue-outlined boxes in the second column need to be multiplied by the blue-filled box labeled the "cost of a square". It is really important to see that we multiply the same way we factored them out.

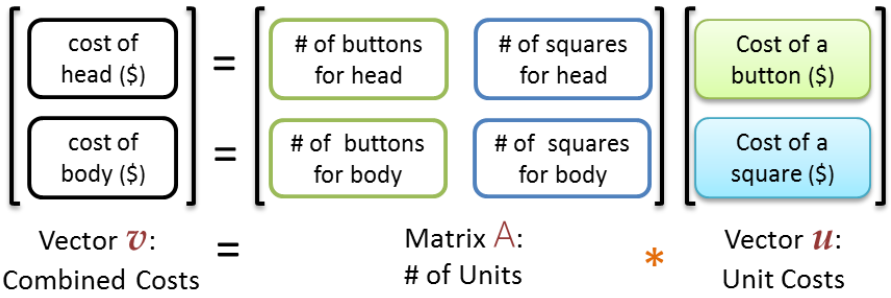


Before we go further, let's look at the matrix form with actual values:

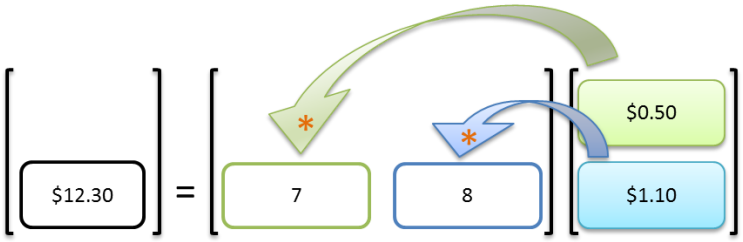


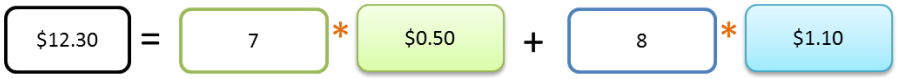
When you look at the above matrix form, you should immediately notice one thing: how the "cost of a button" and "cost of a square" got separated as a *vector* (on the right). At the same time, the resulting matrix in the middle, has all of the "number of units" (i.e., number of buttons and squares of cloth) in its entries. The output vector on the left is the cost vector, and has "combined costs" (or "total costs") resulting from each linear combination.

The most important thing to notice is that how the original two linear combinations (functions) got separated as a product between a matrix that represents "number of units" and a vector that represents "unit costs" (i.e., the cost of a button and the cost of a square). We can express this relationship as $\boldsymbol{v} = A\boldsymbol{u}$, as shown below. Remember, we represent vectors with bold letters to indicate that they are not just numbers, but rather, an ordered set of numbers.



This separation as a product is not surprising given that the original linear combinations calculated the same product. That is, it multiplied the "number of buttons" by the "cost of a button" to get the cost of all buttons (for head or body). Then, it added that with the cost of squares, which is calculated using a similar product. The above matrix-vector product captures both of these linear combinations succinctly. The way we calculate this product (linear combination) for the body of the teddy bear is shown below. This product has a name: **dot product**. In matrix form, the dot product is taken between a row of the matrix and an input vector as shown below. In our example, the dot product has two multiplication and one addition operation (i.e., a linear combination).





As you can clearly see, the dot product is just a fancy name for calculating a linear combination. Thus, "taking the dot product" is essentially "calculating a linear combination". As shown above, the dot product produces one single scalar value, which is $12.30, in this example.

The computer code below shows how to multiply a matrix and a vector. The function accepts the two rows of a 2×N matrix, and an input vector of length N. The first statement in the function creates a new output vector, which will be returned as the output of this function. It has only 2 entries, because the matrix has only 2 rows. The first entry (in fact, the 0th entry, because computers start counting from 0 instead of 1), is evaluated using the dot product between `row1` and the input vector, `vec`. Similarly, the next entry in the output vector is the dot product between `row2` and input vector. Recall that we wrote the function `dotProduct` in the previous chapter in Section 7.10.2.

```
function matrixVectorProduct2xN( row1, row2, vec, N )
{
      output_vec = new Array[2]              // create new output vec

      output_vec[0] = dotProduct( row1, vec, N )
      output_vec[1] = dotProduct( row2, vec, N )

      return output_vec
}
```

```
row1 = [5,6]                                 // row 1 of matrix
row2 = [7,8]                                 // row 2 of matrix
in_vec = [0.50, 1.10]                        // input vector

out_vec = matrixVecProduct2xN( row1, row2, in_vec, 2 )

print( out_vec )                             // prints 9.10 12.30
```

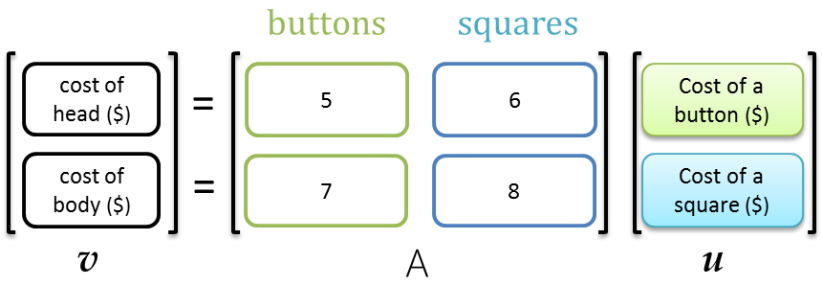
In the second block of the computer code, we call the function we defined. We use the matrix and input vector we used for the teddy bear as inputs to this function. The print statement should print the same costs we saw above. Notice that N is 2 in this example because we had only two columns in the matrix. However, N can be any number, and the length of each row and the input vector must also be N. Notice that we use arrays for rows as well as the input/output vectors, which are actually columns, because you can look at a row or a column as just a vector (a sequence of numbers).

A row or a column of a matrix is a vector

The computer code clearly shows how each entry of the output vector is calculated — as a dot product (a linear combination) between a row of the matrix and an input vector. You should clearly understand this computer code before we move on because it nicely summarizes what we learned so far.

8.3 What Does a Matrix Represent?

Let's take another look at our multiplication of matrix A by the "unit cost" vector, $\boldsymbol{u}$, to produce the "combined costs" vector $\boldsymbol{v}$. Assume that the "unit costs" are variable. That is the cost of a button or the cost of a square (of cloth) can increase or decrease.



To understand what a matrix represents, first, let's look at scalar multiplication function, multiplyByScalar_a(x). Let's define it as:

multiplyByScalar_a(x) $= a * x$ [e.g., $y = 5x$] **(1)**

In function multiplyByScalar_a(x), number a serves as the **scalar multiplier** or the **scalar coefficient** (to multiply any input with). In other words, scalar a (e.g., 5) fixes one input to multiplication. Therefore, the other input always gets multiplied by a. In other words, number a "configures" or "sets-up" the multiplication operation so it behaves in a certain way. Similarly, let's look at a scalar linear combination, or dot product, called multiplyByVector_**a**($\boldsymbol{x}$) as:

multiplyByVector_**a**($\boldsymbol{x}$) $= \boldsymbol{a} \cdot \boldsymbol{x}$ [e.g., $y = 5x_1 + 3x_2$] **(2)**

Note that both $\boldsymbol{a}$ and $\boldsymbol{x}$ are vectors. Vector $\boldsymbol{a}$ is a constant vector. It is the **vector multiplier** (or **vector coefficient**) for input vector $\boldsymbol{x}$. We "multiply" (i.e., take the dot product between) the vector coefficient and the input vector to produce a *scalar* output. Similarly, we can write a function multiplyByMatrix_A. We can define that as:

multiplyByMatrix_A ($\boldsymbol{x}$) = $A\boldsymbol{x}$ [e.g., $\mathbf{y} = A\boldsymbol{x}$] **(3)**

where, $\boldsymbol{x}$ is the input vector. In function multiplyByMatrix_A ($\boldsymbol{x}$), matrix A serves as a **Matrix Multiplier** (or a **Coefficient Matrix**) to "multiply" the input vector with. Notice that as in (2), "multiplying" means taking dot products. However, unlike in (2), we have to take multiple dot products, because output $\boldsymbol{y}$ is a vector. The matrix A provides a set of vector coefficients so we can take dot products with any input vector to produce an output vector. Therefore, matrix A captures our two linear combinations. That's what a matrix represents.

A matrix captures multiple linear combinations

Let's take a minute to notice the similarity between (1), (2), and (3). All three are multiplication operations taking two inputs and producing one output. However, (1) has scalar inputs and a scalar output, (2) has vector inputs and a scalar output, and (3) has one vector and one matrix input producing a vector output. That is, all three are multiplication operations designed for different inputs and outputs.

By itself, a matrix is not a function. However, as we now know, a linear combination is a function. The matrix "configures" linear combinations by providing the multipliers or coefficients. For instance, consider (1):

$f(x) = ax$ or multiplyByScalar_a(x) = ax

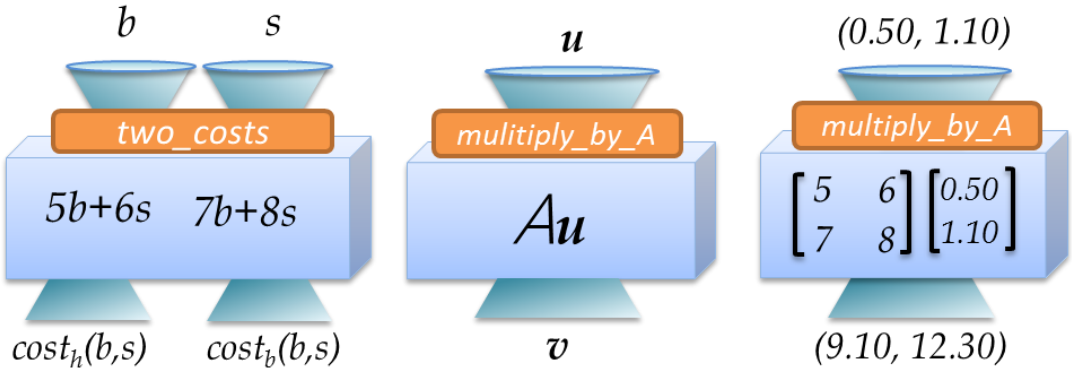
What's the role of a? Value a is the coefficient of multiplication. Graphically, it specifies the slope of the line. Geometrically, 'a' specifies, by how much input x is scaled (or amplified). Not only that. If 'a' is negative, it could cause reflection of input. If 'a' is complex, it could cause rotation. Similarly, matrix A can do similar things to an input, which is a vector in this case. So, we can represent it as:

$f(\boldsymbol{x}) = A\boldsymbol{x}$ or multiplyByMatrix_A($\boldsymbol{x}$) = A$\boldsymbol{x}$

That's what we mean when we say matrix A acts as a multiplier (or a coefficient) for the "multiplication function" (dot products). This is what we modeled as $\boldsymbol{v} = A\boldsymbol{u}$ above where $\boldsymbol{u}$ is the input "unit cost" vector, $\boldsymbol{v}$ is the output "combined costs" vector, and A is the matrix specifying coefficients (numbers of buttons and squares).

The coefficients in the matrix determines the exact multiplication function used for the input vector. Note that, in the context of matrices, the word "multiply" means taking all the dot products (calculating linear combinations). One other caveat: when we name a function multiplyByMatrix_A, it means to use A as the first input into multiplication (i.e., pre-multiply). You will see the importance of this shortly.

The following Visual Models should make the role of a matrix perfectly clear.

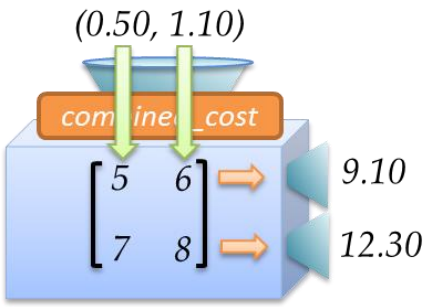


Before we discuss the Visual Model, we should clarify some terminology. A function can accept different types of inputs. It can accept real numbers (scalars) and produce a scalar as output. We call such a function a **scalar**-function. For instance, $f(x) = 5x$ is a scalar function. A function can also accept a vector (a sequence of numbers) as input and produce a vector as output. Such functions are vector functions. Multiplying by a matrix represents such a **vector**-function.

Now, let's start with the first Visual Model on the left. This models our two linear combinations as a single relationship. The scalar variables b and s represent the "unit cost" of a button and a square, respectively. The outputs cost_h and cost_b represent the cost of the head and body, respectively. As you can see, this represents two linear combinations (dot products), since we have two outputs. Therefore, we have two separate recipes: recipe $5b+6s$ to calculate the cost of producing the head, and $7b+8s$ to calculate the cost of producing the body. Although we have two linear combinations, note that both of them depend on the same inputs, b and s. That's why these two linear combinations are related.

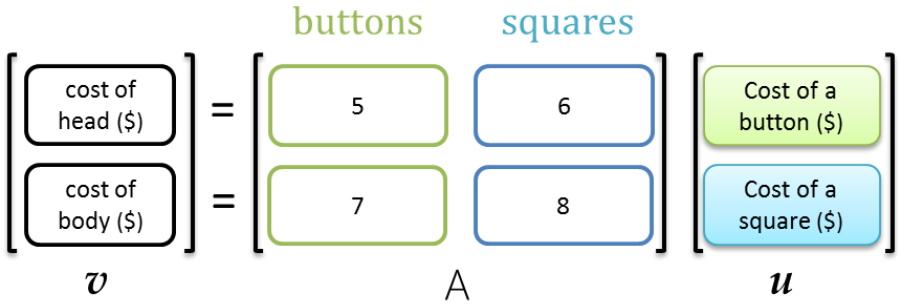
The most important thing to observe here is that the Visual Model on the left is equivalent to the one in the middle, which is the matrix representation of the same two linear combinations. By comparison, you can see that the product $A\boldsymbol{u}$ (matrix A multiplied by input vector $\boldsymbol{u}$) calculates the same outputs (in vector form), and hence represents a vector-function. For instance, to make our teddy bear, we have one linear combination to make the head and another to make the body, using the same inputs (buttons and squares). The equivalent vector-function accepts a vector as its input and produces a vector as its output. It can do so because it contains two separate linear combinations to create each scalar output.

The Visual Model on the right captures our example with numbers for matrix A, input vector $\boldsymbol{u}$ and output vector $\boldsymbol{v}$. Sometimes, it may be easier for you to visualize the entire relationship between input, output, and matrix, if the outputs are shown on the side, instead of on the bottom, as shown with the Visual Model. If you take the dot product between the input and *top row*, you get the *top output*. That is, $5*0.50$ plus $6*1.10$ produces 9.10. If you take the dot product between the input and the *bottom row*, you get the *bottom output*. This is an easy way to remember how to do the actual dot products. Remember, this reflects how we factored out the two linear combinations in the first place, to produce a matrix and a vector.



8.4 Modeling with a Matrix

Let's summarize what we achieved so far. What did we really achieve? We built a cost model for making a teddy bear. That cost model is captured by the simple matrix equation, $\boldsymbol{v} = A\boldsymbol{u}$, where $\boldsymbol{u}$ is the input "unit cost" vector and $\boldsymbol{v}$ is the "combined cost" output vector, as shown below. With this model, we can calculate the costs for making the head and body of a teddy bear, for given "unit costs" (cost of one button and cost of one square).



When modeling the cost of making a teddy bear, matrix A represents how components (buttons and squares of cloth) are combined to make either the head or the body. It specifies that the head needs 5 buttons and 6 squares of cloth (1st row of the matrix), and the body needs 7 buttons and 8 squares of cloth (2nd row of the matrix).

Our matrix A is called a 2×2 matrix since it has two rows (calculates two linear combinations) and two columns (accepts input vectors with two components). In other words, it accepts vectors with two components and produces output vectors with two components, and thus describes two linear combinations of two input variables. A more general matrix is an $m \times n$ matrix, which has m rows (m linear combinations) and n columns (accepts inputs vectors with n components). Therefore, it captures m linear combinations, each with n input variables.

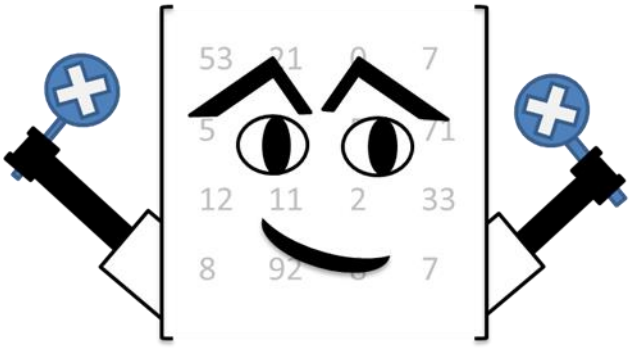
An $m \times n$ matrix captures m linear combinations of n input variables

To summarize, the following figure shows how we use a matrix to model a linear relationship with two outputs and two inputs. As an example, coefficient $_{out1}C_{in1}$ gets multiplied by $input_1$ and contributes to $output_1$. As you can see, coefficients in the 1st column of the matrix get multiplied by $Input_1$, and coefficients in the 2nd column get multiplied by $Input_2$. Similarly, coefficients in the 1st row of the matrix contribute to $Output_1$, whereas coefficients in the 2nd row contribute to $Output_2$. This clearly shows how a matrix is used to model a linear relationship with two outputs and two inputs — that is, a relationship between an input vector and an output vector.

$$\begin{bmatrix} Output_1 \\ Output_2 \end{bmatrix} = \begin{bmatrix} {}_{out1}C_{in1} & {}_{out1}C_{in2} \\ {}_{out2}C_{in1} & {}_{out2}C_{in2} \end{bmatrix} \begin{bmatrix} Input_1 \\ Input_2 \end{bmatrix}$$

We looked at what a linear combination is. We should also keep in mind what it is not. A linear combination cannot do any non-linear operations on its input. For instance, a linear combination cannot square or cube an input. Therefore, a matrix, which describes a set of linear combinations, cannot perform non-linear operations on input.

Now, you should be able to look at a matrix in this new light — a matrix provides multipliers (or coefficients) to linear combinations. The multipliers in the matrix describe how much of each component in the input is used to produce a component in the output.



Matrix Guy:
Supplying Multipliers since 1800's
(family owned & operated)

8.4.1 *Extending Linear Models*

When we examined our function toolbox, the first thing we looked at was the linear function

$$y = a\,x \qquad \text{or} \qquad f(x) = a\,x \tag{1}$$

where x is a scalar variable, a is a scalar coefficient, and y is a scalar output. Compare this with the matrix model we just saw:

$$\boldsymbol{v} = \mathrm{A}\boldsymbol{u} \qquad \text{or} \qquad f(\boldsymbol{u}) = \mathrm{A}\boldsymbol{u} \tag{2}$$

or, if we use the same variables as in (1)

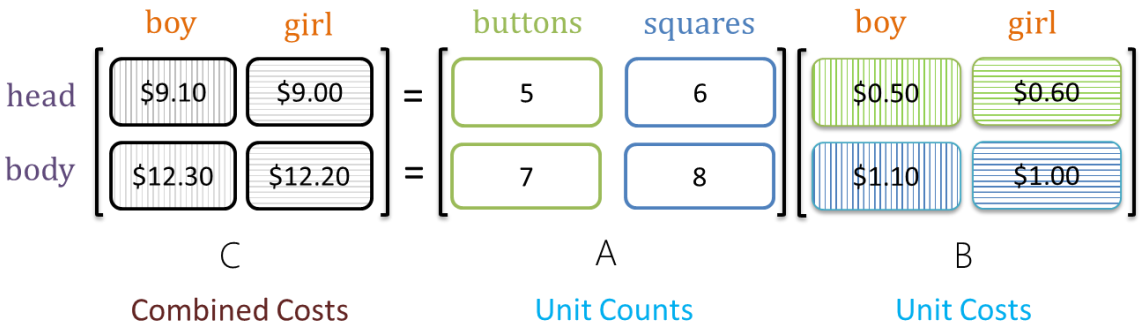
$$\boldsymbol{y} = \mathrm{A}\boldsymbol{x} \qquad \text{or} \qquad f(\boldsymbol{x}) = \mathrm{A}\boldsymbol{x} \tag{3}$$

In (3), the input variable $\boldsymbol{x}$ is a vector, output variable $\boldsymbol{y}$ is a vector and the coefficients (multipliers) are provided by matrix A. What does this tell us? Models (1), (2) and (3) are all linear relationships. Model (1) is a linear relationship between a scalar input and a scalar output. Here, just a simple scalar multiplier (coefficient) does the job. Model (2) is a similar linear model, but the relationship is between a vector input and a vector output. However, a simple scalar multiplier is not sufficient in this case. We need a matrix to supply multipliers for this linear relationship, because this model captures multiple linear combinations — one linear combination for each output component.

8.5 Matrix Multiplication

So far, we learned what a matrix represents and how to multiply a 2×2 matrix by a 2×1 vector (an input with two components). Now, we want to see how to multiply a matrix by another matrix, and what it means.

Earlier, we saw that a matrix can multiply one input vector. Well, if it can multiply one input vector, it can multiply two, three, or any number of input vectors. That's pretty straightforward. One such example is given below. Assume that, instead of one teddy bear, we package two teddy bears, a girl bear and a boy bear (say for Valentine's). We use the same number of buttons and squares for both bears. The only difference is that the buttons and squares (of cloth) for the girl bear have different costs (because of different materials). To represent these different costs, we need to add another input column vector. As a result, our output also ends up with 4 linear combinations — one column for the boy bear and one column for the girl bear. What you have below is one matrix multiplied by another matrix. You need 4 dot products (linear combinations) to calculate the final output.



Can you write the four linear combinations to calculate the output matrix? You already know how to write two linear combinations for the boy bear. You can do the same for the girl bear, except one thing: you have to use the girl column in input matrix B. That's it. Another shortcut people use to remember the order of multiplication is this: say we want to calculate the cost of the head of the girl bear (i.e., entry with $9.00). Which entry is that in the output matrix C? It is at row 1 and column 2. So, you just take the dot product between row 1 of matrix A and column 2 of matrix C. That's it.

The computer code below shows how we can compute the above matrix multiplication using two calls to our `matrixVecProduct2xN` function we defined above. We call that function once to calculate the output column for the boy bear, with matrix A and input vector for the boy as inputs. Then, we call that function again to calculate the output column for the girl bear, with the same matrix A but the input vector for the girl bear. The two output columns are printed, which prints the same values we calculated above. Please make sure you understand this example clearly, because it illustrates how you can easily do matrix multiplication using a computer.

```
row1 = [5,6]                                    // row 1 of matrix
row2 = [7,8]                                    // row 2 of matrix

in_col1 = [0.50, 1.10]                          // input vector 1 (boy)
in_col2 = [0.60, 1.00]                          // input vector 2 (girl)

out_col1 = matrixVecProduct2xN( row1, row2, in_col1, 2 ) // for boy
out_col2 = matrixVecProduct2xN( row1, row2, in_col2, 2 ) // for girl

print( out_col1 )                               // prints 9.10 12.30
print( out_col2 )                               // prints 9.00 12.20
```

Matrix multiplication is really easy as we saw above — it's just taking a bunch of dot products (i.e., calculating linear combinations). Now, let's take one last look at it to understand how to interpret the result we got.

Here is the meaning of each matrix:

- Unit Costs (B): Contains cost of each "unit" (a button and a square). Columns are for boy and girl and the rows are for buttons and squares.
- Unit Counts (A): Contains number of units (buttons and squares) needed. Columns are for the buttons and squares and rows are for the head and the body.
- Combined Costs (C): Contains the combined costs (cost of a head or a body). Columns are for the boy and girl and rows are for the head and the body.

As you can see, two factors of the product (matrix A and B) contain information about a "unit" (buttons and squares) – i.e., how many of them are needed for the head and the body and cost of each for the girl and boy bear. The result of multiplication (matrix C) combines those two to produce the cost of each part (head or the body) of each bear (girl and boy).

The above matrix multiplication is analogous to a scalar multiplication case where we have:

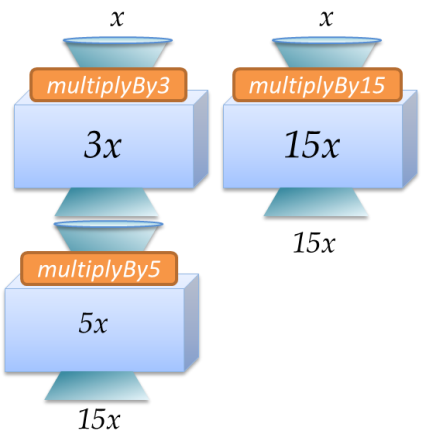
Cost of a bag of candy = Number of candies in a bag * Cost of one piece of candy

The difference in the case with matrices is that, we can track multiple such costs and unit types in a matrix and perform all those cost calculations simultaneously with a single matrix multiplication! That's because matrix multiplication evaluates multiple linear combinations simultaneously. That shows that matrices and their associated operations can represent much richer relationships than simple scalar values and their operations.

Matrix multiplication evaluates multiple linear combinations simultaneously

8.6 Matrix Multiplication as Function Composition

In this section, we are going to look at matrix multiplication as function composition. Before we go there, I want to remind you that scalar multiplication can be thought of as function composition as well. Consider the following function:



$f(\text{input}) = 5 * 3 * \text{input}$

With function composition, the above can be written as:

$f(\text{input}) = \text{multiplyBy5}(\ \text{multiplyBy3}(\ \text{input}\ )\ )$

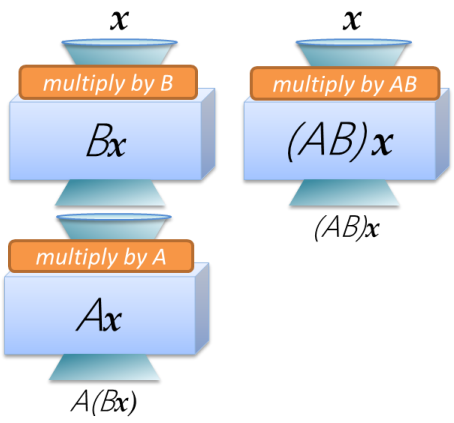
First, we multiply the input by 3, and then, we feed that result into a function that multiplies its input by 5, as shown by the Visual Model. This function composition is equal to:

multiplyBy15(input) = 15 ∗ input

To make this even clearer, the computer code for this composition is given below:

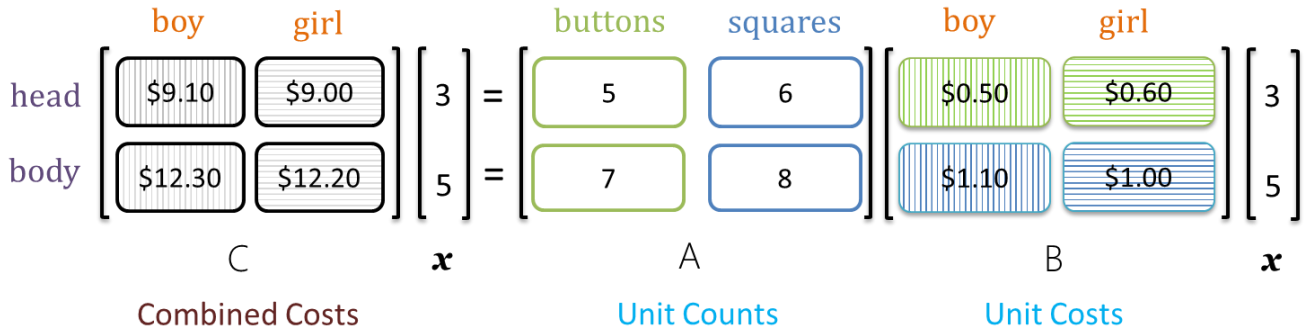
```
function multiplyBy5(input)
{
      result = 5 * input
      return result
}
function multiplyBy3(input)
{
      result = 3 * input
      return result
}
function multiplyBy15(input)
{
      result1 = multiplyBy3(input)
      result2 = multiplyBy5(result1)
      return result2
}
```

We can show that the same function composition applies to the product of matrix A and matrix B, as shown by the Visual Model.

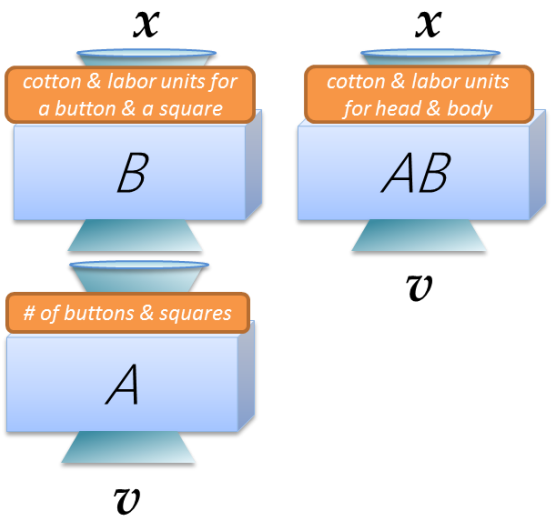


To do that, first, we need to multiply the matrix product AB by an input vector $\boldsymbol{x}$ and note the result. Then, we have to multiply matrix B by the input vector, to produce B$\boldsymbol{x}$, which is a single column vector, and then multiply matrix A by this result B$\boldsymbol{x}$ and compare that with the previous result.

You can easily do that with the girl and boy bear example we looked at before. As shown below, apply input vector $\boldsymbol{x}$ to each side and verify that you get the same result. I leave it as an exercise for you to try. Remember, to apply $\boldsymbol{x}$ first to B when evaluating the right side.



The Visual Model to the right shows how matrix multiplication is equivalent to function composition. If we treat multiplying by a matrix as a function, multiplying by a product of two matrices represents another function, which is equivalent to the composition of two functions.



This has significant implications in matrix algebra. For instance, we know that function composition is not commutative, in general. That is, function f feeding into function g is not equivalent to function g feeding into f. The same applies to matrix multiplication. Matrix multiplication is not commutative either. That is,

$$AB \neq BA$$

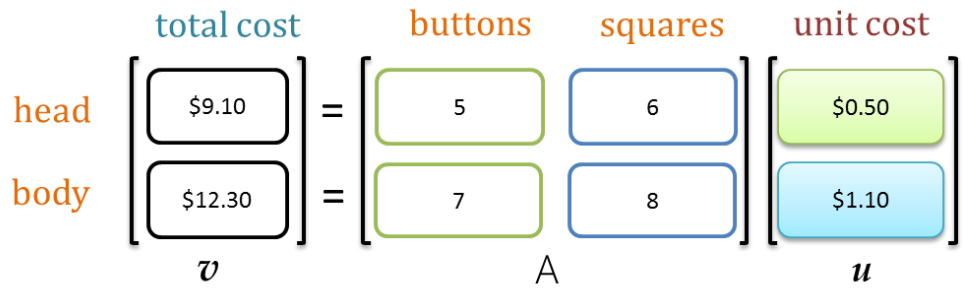
As an exercise, use our matrices A and B to calculate the product BA. You can immediately see that we don't get the same numbers nor, components with the same meaning.

This is one area where matrix multiplication differs from scalar multiplication. As we all know, scalar multiplication is commutative. That is, we can switch the order of two inputs to scalar multiplication operator. At the same time, we showed that scalar multiplication can be thought of as function composition, too. Why does one function composition (scalar multiplication) preserve commutative property while another function composition (matrix multiplication) does not? Here is a hint: matrix multiplication is not really "multiplication". Can you guess now? Matrix multiplication carries out a bunch of dot products, or linear combinations. When you calculate the matrix product AB, the rows of A

get multiplied (linearly combined) by columns of B. When you calculate matrix product BA, the *rows of B* get linearly combined with *columns of A*. As you can see, the linear combinations we form to calculate AB are very different from the linear combinations we form to calculate BA. There is no such a difference in scalar multiplication — you multiply the same two inputs together regardless of their order.

8.7 Row View vs. Column View of a Matrix

Recall how we calculated the total cost of a teddy bear using the model $\boldsymbol{v} = A\boldsymbol{u}$, where $\boldsymbol{u}$ is the input "unit cost" vector, $\boldsymbol{v}$ is the output "total cost" vector, and matrix A represents the number of buttons and squares used for the head and the body.



We used two linear combinations to calculate each component of the output as:

$$9.10 = 5 * 0.50 + 6 * 1.10$$

and

$$12.30 = 7 * 0.50 + 8 * 1.10$$

Or, using dot products, we can write each output as a dot product between each row of matrix A and input vector, as follows:

$$9.10 = (5, 6) \cdot (0.50, 1.10)$$
$$12.30 = (7, 8) \cdot (0.50, 1.10)$$

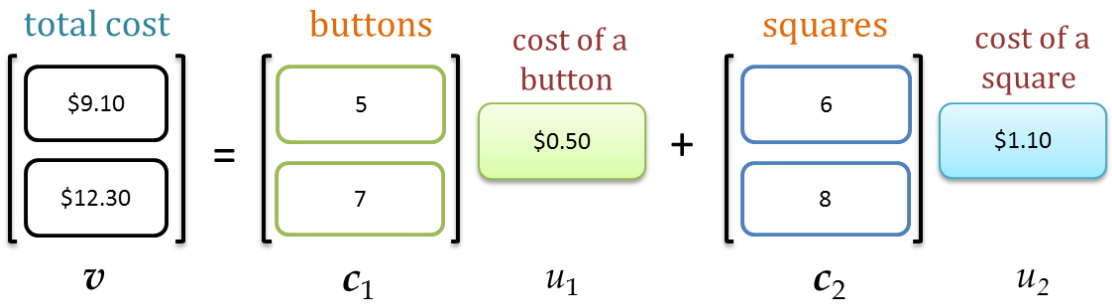
For instance, the row vector (5, 6) is the 1st row of matrix A. The above can be written as:

$$v_1 = \boldsymbol{r}_1 \cdot \boldsymbol{u} \quad \textbf{(1)}$$
$$v_2 = \boldsymbol{r}_2 \cdot \boldsymbol{u} \quad \textbf{(2)}$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the 1st and 2nd *rows* of matrix A, respectively and v_1 and v_2 are the components of vector $\boldsymbol{v}$. Here, we focus on how each component in the output vector $\boldsymbol{v}$ got

produced, as a dot product between a ***row*** of the matrix and the input vector, $\boldsymbol{u}$. This view is called the "*row view*" because we look at each row of the matrix A at a time.

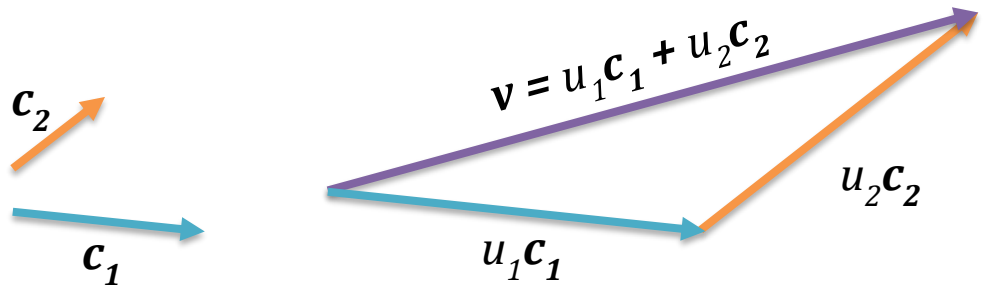
However, there is an alternative way to look at the same model $\boldsymbol{v} = A\boldsymbol{u}$, using columns of the matrix A, as shown below.



In this view, we look at the output vector $\boldsymbol{v}$ as a linear combination of two input vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, where $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are columns of matrix A, as shown above. In other words, we have:

$$\boldsymbol{v} = u_1\boldsymbol{c}_1 + u_2\boldsymbol{c}_2 \qquad \textbf{(3)}$$

Notice that u_1 and u_2 are just scalar components of the input vector $\boldsymbol{u}$, where u_1 is the cost of a button and u_2 is the cost of a square. Notice that when we multiply a column vector by a scalar coefficient, each entry in the column vector gets multiplied by the scalar coefficient. In other words, the column vector gets "scaled", or magnified. Therefore, this "*column view*" tells us the following: each column of the matrix gets scaled by a component of the input vector, and those scaled column vectors are added together to produce the output vector. From a 2D geometric viewpoint, the following figure shows how column vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ get scaled by u_1 and u_2 and added together to produce vector $\boldsymbol{v}$.



Both the row view and column view are equally valid, and produce the same output in the end. Notice that both forms are linear combinations, but (1) and (2) are linear

combinations between scalar components (dot products), each with a scalar result, whereas (3) is a linear combination of vectors, with a vector result.

The "row view" focuses on how *each component of the **output** vector* gets made, whereas the "column view" focuses on how *each component of the **input** vector* contributes to the output vector.

8.8 Linear Transformations

If you have ever played any computer game, you have seen objects on the screen move, grow, shrink, stretch, and rotate, as you play the game. How do you actually do that? If an object is composed of line segments, it becomes a matter of moving, growing, stretching, and rotating these line segments. What's a good model to do that? As you know, we can model line segments with vectors. Great! Now, how do we manipulate these vectors so that they grow, stretch, rotate, etc.? To do that, we need to produce an output vector from an input vector. Luckily, we know one way to do that: matrices. So, let's see how we can do that.

Assume you have a line segment drawn in a 2D Cartesian plane, starting at the origin and ending at coordinates (5, 3). Now, we can use these coordinates as our input vector. It has two components, and hence, the vector is in $\mathbb{R}^2$. Let's say we want to grow (scale) this line segment, in both x and y directions, by a factor of two. To generalize, let's say our input vector is (x, y), and the output vector is (x', y'). So, we can write the two relationships for x' and y' using two inputs x and y:

$x' = 2x$ [scales x input by two]
$y' = 2y$ [scales y input by two]

For instance, if (5, 3) is our input vector, our output vector is (10, 6). Since x' and y' are functions of two variables, x and y, we can generalize the above two equations as two linear combinations:

$x' = 2x + 0y$ [x' is twice of x (and no y)]
$y' = 0x + 2y$ [y' is twice of y (and no x)]

We know how to write these two linear combinations in the matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The above matrix relationship is known as a **linear transformation**, because it's a linear model of the form

$$\boldsymbol{x}' = A\boldsymbol{x}$$

The above is similar to the model $\boldsymbol{v} = f(\boldsymbol{u}) = A\boldsymbol{u}$ that we have encountered before. It "transforms" an input vector to an output vector. So, if you had 10 such input vectors on your screen, what would you do to grow them by 2X in each direction? You guessed right. We transform each of these vectors using our "**transformation matrix**", A. In other words, we multiply matrix A by input coordinates to calculate the output coordinates. This is the basis of a linear transformation.

Notice that we can model an object as a bunch of vectors, where each vector is grounded at the origin. If we transform all of these vectors using the same transformation matrix, we are essentially transforming the entire object.

Quick quiz: what is the transformation matrix you would use if you wanted to stretch an object in x direction by 2, but no change in the y direction? We can model this using the following two linear combinations.

$$x' = 2x + 0y$$
$$y' = 0x + 1y$$

The computer code to do this transformation is given below, using the same matrix multiplication function we used before. Since this code is very similar to previous examples, you should be able to understand it clearly.

```
row1 = [2,0]                                    // row 1 of matrix
row2 = [0,1]                                    // row 2 of matrix
in_vec = [10, 15]                               // input vector

out_vec = matrixVecProduct2xN( row1, row2, in_vec, 2 )

print( out_vec )                                // prints 20, 15
```

What if we wanted to get the same object, without any change? That's basically the following matrix. It is known as the **identity matrix**, because its output is exactly the same as its input. Therefore, this transformation is called the **identity transformation**.

$$x' = 1x + 0y$$
$$y' = 0x + 1y$$

Instead of stretching our line segment, what if we wanted to reflect it around the y-axis? We just have to negate the x coordinate:

$$x' = -1x + 0y$$

$$y' = 0x + 1y$$

What if we wanted to rotate by 180^0? We have to negate both coordinates. Remember, negation is rotation by 180^0.

$$x' = -1x + 0y$$
$$y' = 0x - 1y$$

Similarly, you can perform rotation by any arbitrary angle t counter-clockwise, with

$$x' = x\cos(t) - y\sin(t)$$
$$y' = x\sin(t) + y\cos(t)$$

where $\sin(t)$ and $\cos(t)$ are constant scalar multipliers. In general, in a linear transformation, we produce each output coordinate using input coordinates. Thus, we can write:

$$x' = f(x, y)$$
$$y' = g(x, y)$$

where f and g are linear combinations (linear functions of multiple input variables). Now we know how to model such a set of linear combinations — with matrices.

8.8.1 Performing Multiple Transformations at Once

What if we wanted to do two transformations at once? Well, say we want to stretch our line segment in the x direction by a factor of two and rotate it by 180^0. What we are essentially doing is performing one transformation and then feeding that output into the next transformation. What is this called? It is called function composition. But wait! How do you do function composition with matrices? As we saw before, this is matrix multiplication. Remember, if we have two transformations represented by matrices A and B as follows:

$\boldsymbol{v} = A\boldsymbol{u}$ [first transformation: input $\boldsymbol{u}$, output $\boldsymbol{v}$]
$\boldsymbol{w} = B\boldsymbol{v}$ [feed $\boldsymbol{v}$ into second transformation, to get $\boldsymbol{w}$]

If we substitute for $\boldsymbol{v}$ in the second transformation, we get:

$$\boldsymbol{w} = B(A\boldsymbol{u})$$

We can write this as:

$$\boldsymbol{w} = C\boldsymbol{u}$$

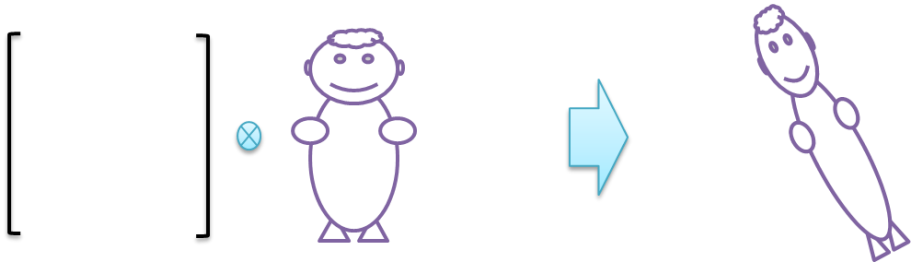
where C = BA.

This is matrix multiplication, as we saw before. This is exactly why we looked at matrix multiplication as function composition. Therefore, if you have to apply multiple transformations to an input object, you can multiply those transformation matrices together to come up with a single transformation matrix capturing all those transformations.

As an exercise, combine a stretching transformation and rotation transformation by multiplying the transformation matrices and verify that you get the combined transformation using the resulting matrix.

This should justify the time we spent learning matrix multiplication. As you can see, matrix multiplication allows us to combine many linear transformations into a single transformation. You can look at it the other way around too. If we were given matrix C above, we could decompose it into two simpler transformations represented by A and B.

The following figure shows how a linear transformation can both stretch and rotate and input image to produce an output image.



Because of this ability to transform objects, matrices are heavily used in computer graphics. Even the font you see on your computer or smartphone screen may be modeled using vectors for the same reason. When you zoom in or increase your font size, you may be doing a matrix transformation.

The next time you play a computer game, or even zoom in a document you read, remember that it is made possible by matrices and vectors.

8.8.2 Multiplying by a Matrix as Two Operations in One

In Section 3.3.2, we saw how scalar multiplication was actually two operations — scaling (amplification) and reflection (rotation by 180 degrees). Matrix transformation captures the same two operations: scaling and rotation. With matrices, however, the rotation can be by any angle in multiple dimensions. Therefore, multiplying by a matrix can do two things to an input vector:

1) Scale the input vector (in each dimension)
2) Rotate the input vector by any angle (in each dimension)

It can be shown that this is all a matrix can do, when we multiply an input vector by a matrix. This is evident from the "column view" of the matrix we learned above. According to that view, an entry in the input vector can only scale the corresponding column vector of the matrix. Scaling each column vector of the matrix produces a set of scaled vectors. Then, we add those scaled vectors together, which amounts to rotation in each direction. The net result is equivalent to scaling with rotation.

As a result, when we multiply an input vector by any matrix, scaling and rotation is all that can happen to that input vector. For instance, you cannot square the input by multiplying by a matrix. That's why we call this a "linear" transformation.

8.9 Solving Linear Models with Multiple Inputs

From section 4.4.2.1, we know how to solve a linear model of one input, like $2x + 3 = 10$. However, what if our linear model has multiple inputs? Many real-world situations lead to linear models with multiple inputs, but just to understand the concept, let's look at a silly problem first.

Let's say there is a store selling fruits, but only 3 kinds — apples, bananas, and cantaloupes. They sell really good fruits at very cheap prices, but they have some weird business practices. First, you can buy any amount of each fruit, but you must buy at least one of each type. Second, the store does not have prices marked for any of the 3 fruits. If you grab some fruits from each type and take them to the cashier, she only tells you the total price. You are really annoyed by this practice (who wouldn't be?). So, you want to find out the price of each fruit. How do you do that?

You come up with a strategy. You buy 3 bags of fruits at 3 different times. You take each bag to the cashier and pay the total price. You know the number of fruits in each bag and the price you paid, so you can model the 3 bags:

2 apples, 3 bananas, 1 cantaloupe ⇨ \$10
3 apples, 2 bananas, 2 cantaloupes ⇨ \$14
5 apples, 3 bananas, 3 cantaloupes ⇨ \$22

Let's say the price of an apple, a banana, and a cantaloupe in dollars is x_1, x_2, and x_3 respectively. Now, we can write an equation for the total price of each bag as follows:

$$2x_1 + 3x_2 + 1x_3 = 10$$
$$3x_1 + 2x_2 + 2x_3 = 14$$
$$5x_1 + 3x_2 + 3x_3 = 22$$

A set of equations like the above 3 equations is called a system of equations. If you look carefully, you should immediately see that each equation is a linear combination. Therefore, they are called a **system of linear equations**.

8.9.1 Solving a System of Linear Equations

When there are multiple linear combinations, now we know exactly how to model such a relationship. If you forgot how we did this for 2 linear combinations, please refer to Section 8.2, where we represented the cost of making a teddy bear. We can model multiple linear combinations with a matrix, with the matrix representing the multipliers (coefficients) and an input vector representing the input variables.

$$\begin{bmatrix} 2 & 3 & 1 \\ 3 & 2 & 2 \\ 5 & 3 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \\ 22 \end{bmatrix}$$

$$A \qquad \boldsymbol{x} \qquad \boldsymbol{b}$$

The above is the matrix representation of the system of linear equations for our fruit puzzler. By taking the linear combinations, you can verify that this matrix equation gives the same 3 equations we have above. This matrix model can be represented succinctly as:

$$A\,\boldsymbol{x} = \boldsymbol{b} \qquad \textbf{(1)}$$

where A is the coefficient matrix, $\boldsymbol{x}$ is the vector representing the price of each fruit, and $\boldsymbol{b}$ is the constant vector representing the total price of each bag.

We have to find $\boldsymbol{x}$, the "unit price vector" of fruits, given $\boldsymbol{b}$, the "total price vector", and the coefficient matrix A, which represents the quantity of each fruit.

How do we find $\boldsymbol{x}$ using the above matrix equation? Conceptually, it is quite easy. How do we solve a linear equation of the form $ax = b$? We express it as:

$$x = b/a,$$

or

$x = a^{-1} * b$, where $a^{-1} = 1/a$

Notice that a^{-1} is the reciprocal (or *multiplicative inverse*) of a. Similarly, we can write (1) as:

$$\boldsymbol{x} = A^{-1}\,\boldsymbol{b}$$

where A^{-1} is the multiplicative inverse of matrix A (which is commonly referred to as just "inverse of A"). However, finding A^{-1}, is a topic for Linear Algebra, a topic that we will not go into. You can find $\boldsymbol{x}$ through a process called Gaussian Elimination, which indirectly evaluates A^{-1}. If you did that, you would find that $\boldsymbol{x}$ = (2, 1, 3). In other words, an apple costs $2, a banana costs $1, and a cantaloupe costs $3. Not exactly cheap as the store would like us to believe.

Whenever we have a linear model with *N* inputs (variables), we need *N* output values to find the inputs that produced those outputs (i.e., to solve for input), as we saw in the above example. Each output value leads to one linear combination. We can model the resulting set of linear combinations using matrices.

This is another example of a real-world use of matrices. Systems of linear equations are quite common in many fields of engineering and sciences. For instance, in electrical engineering, the relationship between the current, voltage and impedance in an entire circuit with a large number of components can be readily modeled using a system of linear equations like the one given above. Similarly, in structural engineering, you can model the forces acting on a connected structure in the same way.

There is one other advantage to expressing a system of linear equations this way: once you express your system of equations as a matrix model, you can readily use computers to solve it. Therefore, even if you haven't heard a word about Gaussian Elimination, if you can model a real-world problem as a matrix equation like the one shown above, you can solve it using a computer. That's the power of modeling. And, now you know how to model a system of linear equations using a matrix. That's what I want to emphasize in this book. Once you have a mathematical model, even a dumb computer can solve it.

8.10 The Story So Far

In the function dynasty, linear combinations play a special role. A linear combination is just a linear function with multiple inputs, producing just one output. However, what if we wanted a linear model that produces multiple outputs (i.e., a vector output) using multiple inputs (i.e., a vector input)? We use matrices to build such models. The whole point of a matrix is to help linear combinations transform an entire input vector to an output vector.

Why do we need all of this? Can't we live without matrices? Remember, the whole point of a function, or a model, is to produce output from input. We need to develop different models to produce different types of output for given inputs. For instance, $f(x) = x$ and $g(x) = 5x$ produce different outputs for the same input, and hence, model two different relationships. Similarly, we need different linear models producing different outputs for the same vector input. To do that, we need matrices. By changing the matrix, we can model different linear relationships.

A matrix provides the multipliers (coefficients) necessary for a linear transformation, which is a function for producing an output vector from an input vector. The same matrix can be used to perform the same transformation on a large set of input vectors. For instance, any set of points in 3D space (e.g., a 3D object) can be transformed to another set of points using a matrix. Any such transformation can perform both scaling and rotation of input vectors. Further, if we want to apply multiple matrix transformations all at once, we can use matrix product to combine all of our individual transformations into a single transformation matrix that captures all of our transformations.

A system of linear equations also models a linear relationship between an input vector and an output vector. Thus, a matrix equation of the form $A\boldsymbol{x} = \boldsymbol{b}$ helps us model those multiple linear equations at once, allowing us to obtain solutions to that system.

I hope this chapter helped you understand the important role that matrices play in modeling real-world relationships.

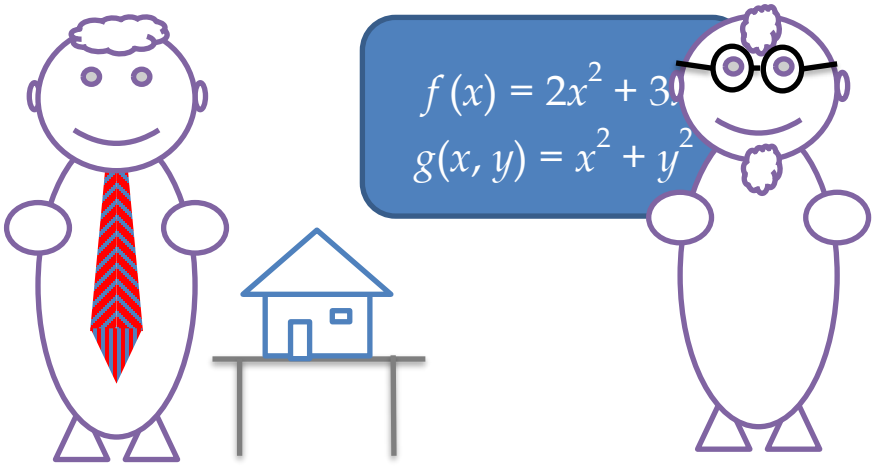
9 Summary: Functions in Perspective

Chapter Overview: This chapter helps you create a coherent view of all the concepts we have seen in this book and how they are related to each other.

Traditionally, math, especially Algebra, is taught as an aggregation of a lot of things, which could seem quite unrelated to most students. It causes students to think, "What's the meaning of all these random things I am learning?", or "Why am I learning about these crazy things called complex numbers and matrices?" I am glad you asked that question, because I am about to reveal …

9.1 The Meaning of Life, err…, Math

Everything we looked at in this book falls under one theme: how we can use math to model the real world. In other words, we look at math as a *tool* for modeling the real world. Just as architects build models of buildings, mathematicians build models of the real world using mathematical relationships (functions).



An Architect's Model vs. a Mathematician's Model

Functions are the centerpiece of this modeling activity. They consume objects as input and produce objects as output. We pick these input and output objects based on what we want to represent, and then pick a suitable model to produce output from our input. So, let's start by looking at …

9.2 Objects

Let's start with the objects that we met in this book, starting with the simplest to the most elaborate:

- Real numbers (scalars)
- Complex (rotated) numbers
- Vectors
- Matrices

A real number can model a 1D (scalar) object, with just one property. A complex number can model a 2D object (a rotated object) with two properties. A vector can model an nD (n-dimensional) object, with n properties. A matrix can model an $m \times n$ dimensional object, which has m vectors, each with n properties.

Now, you should see how these objects are related to each other. For instance, a real number is a degenerate (simpler) case of a complex number, which is, in turn, a simpler case of a vector, which in turn is a simpler case of a matrix. For instance, a vector with one component is a just a real number. Conversely, three real numbers can be represented as one 3D vector (a vector with 3 components). Thus, an algebraic vector is a generalization of a single real number. Similarly, a matrix consists of multiple algebraic vectors (multiple vectors in component form), and therefore, is a generalization of a vector.

For modeling the real world, we have to pick the right object for the right job. For instance, to represent a scalar quantity like height, slope, distance, temperature, etc., a real number is sufficient. However, to represent a property like electrical impedance, we need a complex number that can represent a magnitude and a direction in 2D. To represent n different skills of a person, we need an n-dimensional vector. To represent n different vectors (e.g., to model how n different skills are valued in m regions), we need a matrix. Is the matrix the ultimate object? Not quite. If we continued this extension further, you would meet Tensors. Although we would not describe them in this book, you now have the required intuitive foundation to explore them.

Now that we know what the above objects can represent, let's look at …

9.3 Functions

Functions are the workers or artisans in the mathematical world. They produce output from input. In other words, a function models the relationship between input and output.

However, all functions are not created equal. Some are simpler than the others. One such simple family of functions is known as …

9.3.1 LINEAR MODELS

Out of all of our models, one of the simplest models is the linear function with one linear term:

$$y = ax \qquad \textbf{(1)}$$

We looked at this function in Section 4.1.1. This linear term has only 1 input variable. What's the analogous linear model, when we have multiple input variables? Yes, it's a linear combination of scalar inputs, which can be also modeled as a dot product. A dot product with just two scalar variables looks like:

$$y = a_1 x_1 + a_2 x_2$$

$$= \boldsymbol{a} \cdot \boldsymbol{x} \qquad \textbf{(2)}$$

In the above linear combination, we take the dot product of two vectors: a constant vector $\boldsymbol{a} = (a_1, a_2)$, and an input vector $\boldsymbol{x} = (x_1, x_2)$, which yields the scalar result y. Remember, vectors $\boldsymbol{a}$ and $\boldsymbol{x}$ can have any number of components, in general. To summarize, a linear combination between two vectors, or a dot product, models a linear relationship between input and output, when the input is a vector and the output is a scalar.

What if we want a linear model that produces a vector output for a vector input? We have it covered too. That's where matrices come in. Why? In (1), we just needed one scalar multiplier (coefficient) to model a linear term. In (2), we needed a sequence (or a vector) of scalar multipliers (coefficients) to model the linear combination. If we have to produce a vector as output, we need one linear combination for *each output component*. Therefore, we need multiple vectors of scalar multipliers (coefficients) to model this relationship. An object with multiple vectors is represented by a matrix. Therefore, we can express such a relationship as:

$$\boldsymbol{y} = \mathrm{A}\boldsymbol{x} \qquad \textbf{(3)}$$

Note that both $\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors, and hence written, with bold symbols. A is a matrix. Compare (3) with (1) and (2), and you can immediately recognize the similarity in their form. All three are some form a product between an input and a coefficient, which is the hallmark of a linear relationship.

All three models given above are linear relationships with different inputs and outputs, and they are summarized in the following table:

	Single Output (R) (Scalar Output)	Multi-Component Output (R^n) (Vector Output)
Single (Scalar) Input (R)	$y = a\,x$ **linear term** (scalar multiplier a)	$\boldsymbol{y} = \boldsymbol{a}\,x$ linear scaling of a vector (vector multiplier $\boldsymbol{a}$)
Multi-component (Vector) Input (R^n)	$y = \boldsymbol{a} \cdot \boldsymbol{x}$ **dot product** (vector multiplier $\boldsymbol{a}$)	$\mathbf{y} = \mathrm{A}\boldsymbol{x}$ **linear transformation** (matrix multiplier A)

As the above table shows, when we have a scalar input, x, we get our simple linear function with one linear term:

$$y = a\,x \qquad (1)$$

producing a scalar output, y. Here, coefficient 'a' is a ***scalar multiplier***. If we need to produce a vector output with a scalar input, we need a ***vector multiplier***, leading to model

$$\boldsymbol{y} = \boldsymbol{a}\,x \qquad (2)$$

which simply produces different scaled versions of the constant vector $\boldsymbol{a}$. This model always produces a vector parallel to vector $\boldsymbol{a}$, so it is quite simple.

When we have a vector input, we have a linear combination between the components of the input vector $\boldsymbol{x}$ and the components of the constant vector multiplier $\boldsymbol{a}$, producing scalar output as given by $y = \boldsymbol{a} \cdot \boldsymbol{x}$. Remember, this is called a **dot product** (scalar product) and can be expanded as

$$y = a_1x_1 + a_2x_2 + \ldots \qquad (3)$$

where coefficients a_1, a_2, etc. come from constant coefficient vector $\boldsymbol{a}$, and scalar input components x_1, x_2, etc. come from the input vector $\boldsymbol{x}$.

If we want a linear relationship that produces a vector output $\boldsymbol{y}$ from a vector input $\boldsymbol{x}$, we have the model $\boldsymbol{y} = \mathrm{A}\boldsymbol{x}$, where A is a matrix. Therefore, A is a **matrix multiplier** in this linear relationship. As we saw in Chapter 8, this is just multiplying a matrix by an input vector. This model is sometimes referred to as a **linear transformation**, or a linear-map.

However, if we expand $\boldsymbol{y} = A\boldsymbol{x}$ using the column vectors of A, which can be written as $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, etc., we get the following **linear combination of vectors** with components x_1, x_2, etc. as our scalar coefficients.

$$\boldsymbol{y} = \boldsymbol{a}_1 x_1 + \boldsymbol{a}_2 x_2 + \ldots \qquad \textbf{(4)}$$

Compare (3) and (4). Both are linear combinations. However, (3) is a linear combination of *scalar components* (a scalar product or dot product), whereas (4) is a linear combination of *vectors*. Whenever we have a vector input, we use a linear combination to produce a linear relationship.

To examine this relationship further, we can look at (1), (2), and (3) as special (degenerate) cases of (4). We can look at (3) as a degenerate case of (4), where each vector $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, etc. has just one element. We can take this observation a little bit further. If you look at (2), which models $\boldsymbol{y} = \boldsymbol{a}x$, it is just a degenerate case of (4), with just one term $\boldsymbol{a}_1 x_1$. Similarly, if you look at (1), which models $y = ax$, it is a degenerate case of (3), with one term, $a_1 x_1$.

What do all these observations tell us? They tell us that all of these models are linear combinations, in some form or another.

Every linear model is a form of a linear combination

As another interesting point, notice that how the multiplication function (operator) is used when the input is a scalar, but how multiplication gives way to linear combinations when the input is a vector. A linear combination can be thought of as a sum of simpler linear models. For instance, each term in (3) and (4) is a simpler linear model.

You may have noticed that the general linear model has a constant term as well. For instance, the linear function $y = ax + b$ has a constant term. This model is still a linear combination. For instance, we can get this linear model from (3), if we have $x_2 = 1$ (i.e., the 2nd input is a constant input). Then we get:

$$y = a_1 x_1 + a_2,$$

which is same as

$$y = ax + b$$

Similarly, we can extend our other linear models, as given in the following table:

	Single Output ($\mathbb{R}$) (Scalar Output)	Multi-Component Output ($\mathbb{R}^n$) (Vector Output)
Single (Scalar) Input ($\mathbb{R}$)	$y = ax + b$	$\boldsymbol{y} = \boldsymbol{a}x + \boldsymbol{b}$
Multi-component (Vector) Input ($\mathbb{R}^n$)	$y = \boldsymbol{a} \cdot \boldsymbol{x} + b$	$\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b}$

Notice that, models with vector outputs have an additional *constant vector* as an offset. Models with a scalar output have an additional *constant scalar* as an offset. We have essentially added another input and another coefficient to each model, with one important detail — this input is always 1. This constant input "shifts", "offsets", or "biases" the model. For instance, when we have $y = \boldsymbol{a} \cdot \boldsymbol{x} + b$, this is the same model as $y = \boldsymbol{a} \cdot \boldsymbol{x}$, if vector $\boldsymbol{a}$ had an extra component, and input vector $\boldsymbol{x}$ had an extra component equal to 1. Similarly, $\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b}$ is same as $\boldsymbol{y} = A\boldsymbol{x}$, if matrix A had an extra column and the input vector $\boldsymbol{x}$ had an extra row (with entry 1).

Therefore, all of the above linear relationships are linear combinations in one form or the other. If you happened to look at each of these models as a separate concept, now you know how to unify them under one umbrella.

However, what do we do when a linear model is insufficient to model the problem at hand? That takes us to …

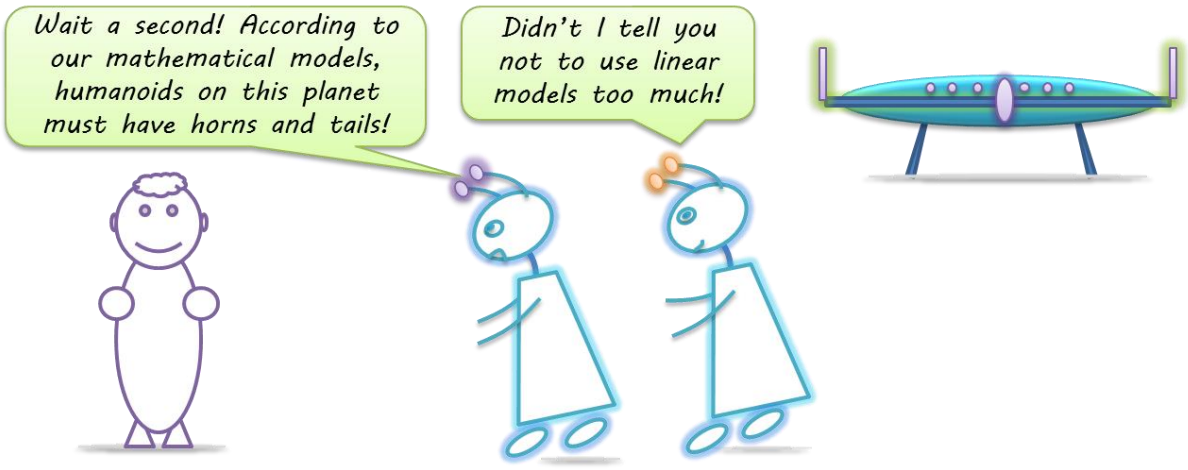
9.3.2 Non-Linear Models

The table below shows the *general* models for the corresponding *linear* models we saw in the previous two tables. For example, the general model accepting an input vector $\boldsymbol{x}$ and producing a vector output $\boldsymbol{y}$ is a vector field. The linear model doing the same thing is just a linear transformation of the form $\boldsymbol{y} = A\boldsymbol{x}$. Notice that the example functions for vector inputs (and output) are given for 3 dimensional vectors — i.e., $\boldsymbol{x} = (x_1, x_2, x_3)$.

	Single Output (R) (Scalar Output)	Multi-Component Output ($\mathbb{R}^n$) (Vector Output)
Single (Scalar) Input (R)	$y = f(x)$ **Scalar Function**	$\boldsymbol{y} = f(t)\,\mathbf{i} + g(t)\,\mathbf{j} + h(t)\,\mathbf{k}$ **Vector Valued Function**
Multi-component (Vector) Input ($\mathbb{R}^n$)	$y = f(x_1, x_2, x_3)$ $y = f(\boldsymbol{x})$ **Scalar Field**	$\boldsymbol{y} = f(x_1, x_2, x_3)\,\mathbf{i} + g(x_1, x_2, x_3)\,\mathbf{j} + h(x_1, x_2, x_3)\,\mathbf{k}$ $\boldsymbol{y} = f(\boldsymbol{x})\,\mathbf{i} + g(\boldsymbol{x})\,\mathbf{j} + h(\boldsymbol{x})\,\mathbf{k}$ **Vector Field**

You should be able to readily distinguish a linear model from a non-linear model (a general model). For example, if we are talking about functions accepting a single scalar variable and producing a scalar output, $f(x) = 3x^2 + 1$ is a non-linear function whereas $f(x) = 5x$ is a linear function. Similarly, if we are talking about functions with multiple scalar inputs producing a scalar output, $f(x, y) = x^2 + y^2$ is a non-linear scalar field whereas $f(x, y) = 5x + 9y$ is a linear combination (linear version of a scalar field). Similarly, if we are talking about a function accepting vector inputs and producing vector outputs, $f(x, y, z) = 5x^2\mathbf{i} + xy\mathbf{j} + z\mathbf{k}$ is a vector field, whereas $f(\boldsymbol{x}) = A\boldsymbol{x}$ is a linear transformation (linear version of a vector field).

Why do linear models get special treatment? There is one simple answer: simplicity. If we can model a problem with a linear model, it is always much simpler than modeling with a non-linear model. Therefore, whenever possible, we try to build a linear model, even as an approximation, because it is much simpler to deal with compared to a non-linear model. However, sometimes, a linear model may not match reality. No matter how hard we try, we cannot model radioactive decay using only a linear model. In such cases, we need non-linear models.



The above discussion should give you a cohesive view of functions. Whenever you meet a new function, try to fit that function into one of the models described here. First, look at its input and outputs. Are they scalar or vector? Is the model linear or non-linear? Answers to these questions should help you categorize the function properly, and get an intuitive feel for how it behaves.

This summary of the linear and non-linear functions brings us to the end of …

9.4 The Story

This is the story of the function dynasty: a multi-generational family in the business of producing models. We started by wondering how functions came into existence. They arose out of sheer necessity: our desire to model the real world. These models could be for simple relationships, from density of objects to really intricate models like the theory of general relativity. All these models are made possible by the members of the function family.

At a basic level, all functions do the same thing: they consume input and produce output. If that's all they can do, how can they become so sophisticated? One property that makes functions so powerful is their ability to consume another member's output, producing more elaborate models. This is known as function composition, and though we fail to realize it, function composition is used heavily in creating intricate models from simple models. This ability makes functions quite versatile as a family, because they can collaborate by feeding one's output into the input of another.

A function usually comes with a counterpart, who can undo what the function does. However, this counterpart, called the inverse of a function, may accept to undo only a part of the output of the original function. Still, this ability to go from an input to output, and

back from output to input, makes functions even more useful. For instance, once we define a function for multiplication, we get its inverse, division, naturally. Therefore, these inverse functions are quite helpful in solving a common problem that functions face: figuring out the input that produced a given output.

The whole purpose of life for a function is to consume input and produce output. The objects that they consume and produce can be as simple as a whole number or as intricate as a vector or a matrix. These objects could represent everyday items, such as a pair of shoes, to models that can transform objects in N-dimensional space.

Some members of the function family are used heavily in modeling. When it comes to modeling, linear functions always enjoy a special place due to their simplicity. We have met linear models, from those that accept only scalars to ones that accept vectors. Their more upscale cousins, non-linear models, allow us to more faithfully model the real world. Our function toolbox contains rich non-linear models of the real world, including Power, Polynomial, Factorial, Trigonometric, Exponential, and Logarithmic models.

Due to this versatility, functions act as a central pillar in many fields. In particular, functions allow us to tie two different fields together — math and computer programming. Functions are the underlying tool in computer programming for building computer models. Since most real-world models are developed today using computers, functions play an integral part in our everyday lives whenever we use a computing device.

The function that fascinated us the most throughout this book is multiplication. Remember, every operator is a function and so is multiplication. One input to multiplication, also referred to as the *multiplier*, determines exactly what happens to the other input. A positive scalar multiplier (a positive number) just scales (amplify) the other input. A scalar multiplier (a real number) can perform both scaling and reflection (rotation by 180^0). A complex multiplier (a complex number) can perform both scaling and rotation by any angle (in a 2D plane). A matrix multiplier can perform scaling and rotation of any input vector of N-dimensions, performing a transformation in N-dimensional space. Notice that, with vectors, multiplication becomes linear combinations, which are sums of simpler linear terms. A matrix multiplier can transform an N-dimensional input vector to an M-dimensional output vector.

Remember, multiplication doesn't behave this way by accident. As any other function, it is a function (operator) we defined, although it may appear to you as it existed ever since the Big Bang. First, we defined multiplication as a function to scale whole numbers. Once we encountered negative numbers, we extended it so that multiplication can understand "sense", and hence, can reverse the sense of an input when needed (i.e., can cause reflection). Once that resulted in complex numbers, we again expanded its definition so

multiplication can rotate objects in a 2D plane. We kept on extending this definition because scaling and rotating are fundamental operations we would like to have in our toolbox since we need them so often in the real-world. Therefore, once we encountered vectors, we needed a function that could do the same thing that multiplication does for real and complex numbers. Multiplication could not be trivially extended to serve this purpose. Therefore, we had to invent a new function: a linear combination, which is a sum of linear terms. That did the job for vectors and matrices. Now, we can use matrices to scale and rotate more complex objects like vectors.

From scalar multiplication, we get scalar division, as the inverse function of multiplication. By multiplying input by itself, we get power functions. From inverse of power functions, we get square roots, cube roots, etc. A sum of power functions produces polynomials, which can be thought of as products of linear functions (factors).

One way to realize the importance of multiplication is to see what kind of algebraic relationships we can build if we did not have multiplication — i.e., if we only had addition. Then, you would only be able to build models like $f(x) = x + x + 3$, which are only a subset of linear models (linear models with integer coefficients). All other algebraic functions, like power functions and polynomials, are made possible by multiplication. That makes multiplication vital. Along with addition, multiplication can be placed at the root of the function dynasty for algebraic functions.

As fundamental as they may be, multiplication and addition alone cannot give rise to all functions we met. There is another important class of functions, known as Transcendental functions (e.g., Trigonometric, exponential, logarithmic) that are indispensable for modeling the real world. These functions help us model rapid growth, rapid decay, waveforms, and rotation with stable, growing, and decaying amplitudes, among many other things.

Although algebraic and transcendental functions are two separate families, they are not isolated as it first seems. The infinite series, which is basically another function (sum) with an infinite number of power terms, is the bridge between them. In practice, especially, when we use computers, transcendental functions are approximated using algebraic functions (a limited number of algebraic terms of an infinite series). For all practical purposes, that places multiplication and addition back at the root of the family tree.

However diverse they may seem, all of these generations of functions are unified by a common purpose: to help us model the world around us.

That's the story of the function dynasty.

EPILOGUE

Algebra is one of the pillars supporting many disciplines in science, engineering, and mathematics. By now, you should have a cohesive and intuitive view of algebra. For instance, now, you should be well prepared to understand an algebraic model in physics, statistics, computer science, or any other field. You should be able to break down any complex mathematical model you meet into the simpler models we discussed here. Now you should be well equipped to learn the concepts in calculus.

If you were mechanically manipulating symbols without much thought, I hope this book helped you build an intuitive understanding about the models we use in math. To achieve that end, we took a very different approach to algebra than many textbooks do.

Rather than looking at functions as an abstract mathematical concept, which maps a domain to a range, we looked at numerous functions you meet in everyday life. Rather than drawing a number-line and defining numbers on that as real numbers, we painstakingly explored how we were forced to expand the notion of numbers, if we start with whole numbers. Rather than looking at a bunch of unrelated functions, we built up each function family, explored relationships among them, and connected them to their inverse and reciprocal cousins. Rather than just drawing the graph of the exponential function or the sine function, we built them up with analogies of pushup-routines and Ferris wheels that showed us how these functions can arise out of familiar activities. Rather than writing down a complicated model for something like the Gaussian distribution, we looked at how we can build one from simpler models using function composition. Rather than just giving the Taylor series expansions of transcendental functions, we gradually built up power series from polynomials using the decimal number system as an example. Rather than saying that a complex number is a number with a real part an imaginary part, we examined what forced us to come up with complex numbers, what they really represented, and their connection to vectors. Rather than drawing an arrow on a Cartesian grid and calling that a vector, we looked at vectors as geometric objects, and as algebraic objects, finally uniting them with linear combinations. Rather than giving a textbook definition of the dot product and the cross product between vectors, we took great pains to explore what they really meant using multi-dimensional skill sets and rotating doors as examples. Rather than introducing a matrix as just a grid of numbers, we spent a lot of time to figure out how linear combinations led to matrices, and what role matrices play in extending linear models to ones that can consume and produce vectors.

That's not all. Rather than giving textbook definitions, we looked at fascinating examples and analogies for negative multiplication, roots, parametric functions, function inverses,

composition, linear combinations, and vectors. Rather than treating subjects like vector fields, linear combinations, vector valued functions, and scalar fields as disjoint concepts, we unified them under one umbrella, showing their similarities and differences. Rather than looking at algebra as endless manipulation of symbols and formulas, we examined its practical connection to various real-world models in physics, and expressed many such models using computer code.

Math books are usually filled with lots of formalism. We prioritized intuition over formalism. Having developed the right intuition, you can get formal definitions of math concepts from any textbook, or even from Wikipedia. However, you need to develop an intuitive understanding to really appreciate both the beauty and utility of math. Providing that intuition is the sole objective of this book.

I hope you enjoyed reading this book. You can obtain the latest copy of this book at reintro2algebra.com. If you notice any errors, omissions, typos, or have any comments or suggestions, please share them by writing to reintro2algebra@gmail.com.

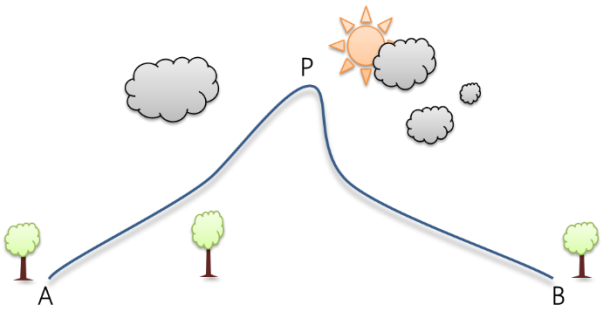
Ruchira Sasanka

SUPPLEMENT: PRELUDE TO CALCULUS

This supplement describes how to look at differential and integral calculus using the intuitive framework we developed in the previous chapters.

OUTPUT VS. CHANGE IN OUTPUT

In the previous chapters, our main focus was to look at a function as a device for producing output from a given input. Sometimes we need to go a step further and understand *how functions behave when we **change** input*.



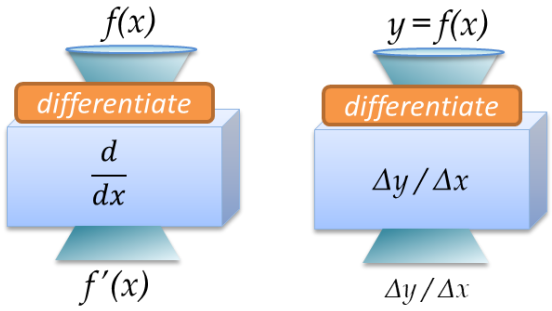
Imagine you want to hike a mountain and there are two trails to the top starting from the same elevation, as shown on the right. One trail (AP) goes up gradually, but the other trail (BP) has a steep section where you have to climb hundreds of feet vertically. Which one would you choose if you want to get to the top with the least trouble? Of course, the one with gradual ascent. What made up your mind? It was not the height (elevation) at a given point on the trail but the impassable steep section. That's an everyday example where we pay attention to the change of a property (elevation in this case) rather than the value of that property at a given point. We can model a trail as a function that outputs an elevation for a given distance from the start (input). In this mathematical model too, we should be able to identify how fast the output changes, if we want to avoid impassable sections.

In many models we build about the real world, we need to know about how output changes in addition to the absolute output value for a given input. In particular, we are interested in how fast the output changes with respect to input (i.e., *the rate of change*). For instance, when you are driving, your speed is determined by how fast your distance (from a reference point) changes, and your acceleration is determined by how fast your speed (or more precisely, velocity) changes. If your speed is too high, you can get a speeding ticket and if your acceleration is too high, your tires may screech or you could lose control. That's why you cannot simply ignore how fast the output changes.

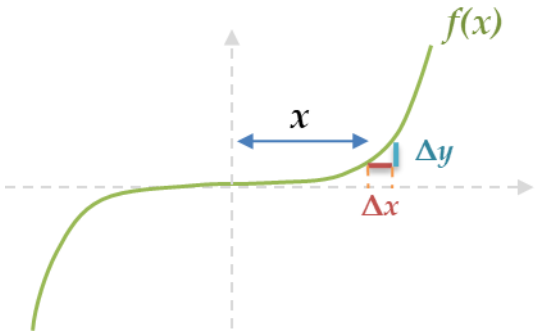
The real-world models we build are often dependent on rate of change. For instance, in Newton's famous second law of motion, $F = ma$, force depends on acceleration, which is the rate of change of velocity. Forces are everywhere in our lives. Therefore, rates of change are fundamental to our existence and the mathematical models we build. Life without

change is boring and if we want to model anything interesting in the real world, we have to incorporate change into our models.

Given that we need to model change, we need operators (functions) that can calculate how fast the output changes in a given function. Such operators are known as the **differential operators**. The output of such an operator is known as the **derivative**.



The differential operator accepts a function as an input and produces another function (derivative) as output. Recall from section 1.10 that operators can do that easily. The differential operator is often written as $\frac{d}{dx}$, when it is applied to a function with input x (e.g., $\frac{d}{dx}[\, f(x)\,]$), as shown in the left image of the Visual Model. However, you should look at it as an operator name, like d_over_dx( ) or D_x( ). For instance, if $f(x) = 3x$, we can apply the differential operator as $\frac{d}{dx}[\, 3x\,]$, or d_over_dx($3x$) or D_x($3x$). The output function is named as $f'(x)$ (i.e., f prime) as shown in the Visual Model. Again, the apostrophe is part of the name and you should think of it has f_prime(x).



At a given input value of x, to calculate the derivative, we change the input by a small amount (Δx) and calculate the change in output (Δy) as shown on the right. A small change is usually denoted by Δ (delta), so delta Δx signifies a very small (infinitesimal) change in input at input x and Δy signifies the corresponding change in output. The derivative is the *ratio* between Δy and Δx. The right image of the Visual Model above captures this view of the differential operator.

The ratio Δy / Δx represents the **slope** of the curve (more precisely the slope of the tangent) at input x. As an example, for the linear model $f(x) = 3x$, which has a slope (gradient) of 3 for all input values, we get $f'(x) = 3$, which is a constant function. In other words, for $f(x) = 3x$, for any given input, the output increases at a rate of 3 – i.e., when you increment x by Δx, output rises by $3 \times \Delta x$.

The following computer code shows `getDerivative` function that numerically evaluates the derivative of a given function '`func`' at any given input `x`. Notice that in programming, a

function (like `getDerivative`) can accept another function (like `func`) as an input argument.

In `getDerivative`, we define `deltaX` (Δx) to be 0.0001 as an example of a small value. Then, first we calculate the output value of `func` at input `x` and then at input `x + deltaX`, allowing us to calculate the derivative, which is defined as:

`[ func( x + deltaX ) – func( x ) ] / deltaX.`

Notice that `getDerivative` calculates the derivative only for a single input value x. If we need to calculate the derivative for a sub-domain of `func`, we need to call it for all points of that sub-domain. That is achieved using a `for`-loop in the last box. There, we evaluate the derivative for x values between 10 and 20. Note that this "for loop" species a step size so that in each iteration, x is incremented by `stepX` (rather than the default step size of 1) giving us x values such as 10.0, 10.0005, 10.001, 10.0015, … etc. (since `stepX` =0.0005). We use the function defined in the middle box (`myFunc`) as the function to find the derivative of. We can use any expression in that function. Currently, we have a quadratic polynomial as the function to differentiate. In the last box, after we find the derivative (`y_prime`), we can print x and `y_prime` and use that to generate a plot (e.g., by using a spreadsheet). This is how your scientific calculator calculates the derivative of a given function and draws it on screen.

```
function getDerivative( func, x )          // finds derivative at x
{
      deltaX = 0.0001                      // small delta value, Δx

      out_x = func(x)                      // output at input x
      out_x_delta = func(x + deltaX)       // output at input x+Δx
      deltaY = out_x_delta – out_x         // Δy = output difference
      slope = deltaY / deltaX              // slope = Δy/Δx
      return slope
}
```

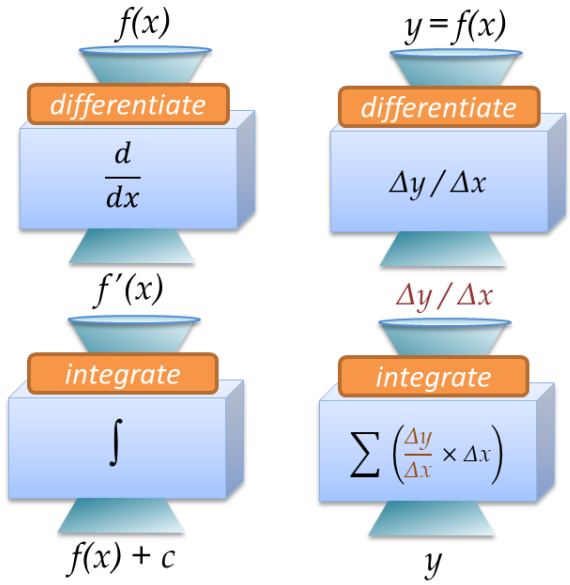
```
function myFunc( x )                       // function to differentiate
{
      return 3*x*x + 2x + 1                // 3x²  + 2x + 1
}
```

```
stepX = 0.0005                             // small step value

for x=10 to 20, stepX                      // for loop with step size
{
      y_prime = getDerivatie(myFunc, x)    // y' = d/dx[ myFunc(x) ]
      print( x, y_prime )                  // print (x, y')
}
```

Notice that since $f'(x)$ is a function, we can reapply the differential operator to it to produce $f''(x)$, which is called the 2nd derivative. We can extend this process to 3rd, 4th, and nth derivative. Can you extend the above computer code by adding another function, `get2ndDerivative()`, to generate the 2nd derivate of a given a function and input value? Hint: you always need two input values to calculate a difference, so you need to calculate $f'(x)$ and $f'(x+\Delta x)$ in order to calculate the 2nd derivative, $f''(x)$.

Differentiation and Integration as Inverse Operations

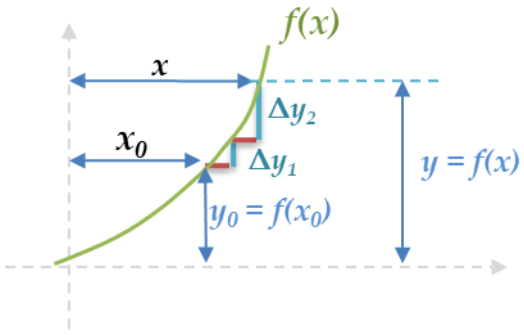
Remember that whenever we talked about an operator before, we also talked about its inverse – because for a given function, we also need to find the input for given output as we discussed ad nauseum in Section 2.3. That's true with the differential operator too. The inverse operator of the differential operator is the **integral operator** (and vice versa). The integral operator is usually denoted by ∫ sign[4]. Again, it would be more intuitive to write it as integral($f(x)$). This inverse relationship is shown in the left image of the Visual Model. Notice that the integral actually produces a family of functions labeled as $f(x) + c$, where c is a constant. This is because if we input function $f(x) + c$ as an input to the *differential* operator, it still produces the same output $f'(x)$, because a constant has a slope of zero. Since the differential operator produces the same output for a family of input functions, the integral operator outputs the same input family.



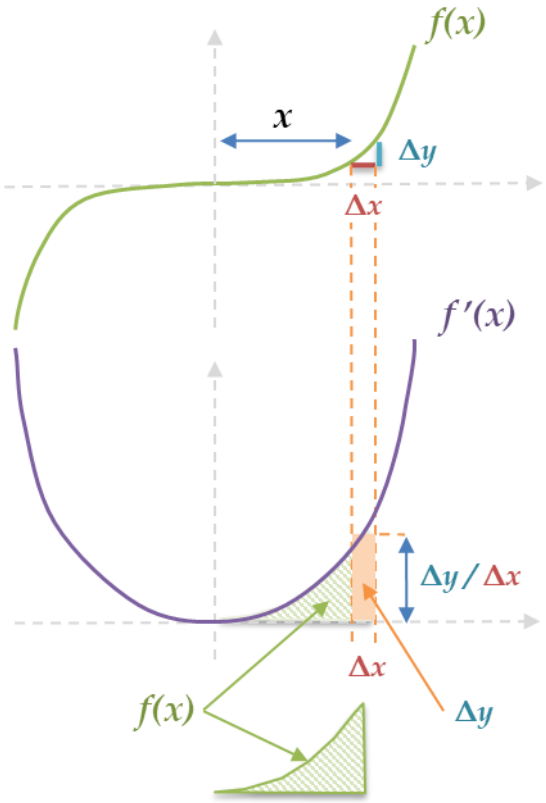
As we know, the differential operator calculates $f'(x)$, the rate of change of output at input x, by *dividing* the change in output by the change in input (i.e., $\Delta y/\Delta x$). Therefore, the integral operator, being the inverse of differential operator, calculates the original change in output (Δy) by *multiplying* $f'(x)$, which is $\Delta y/\Delta x$, by the change in input (Δx) for the same input x, as shown on the right Visual model above. As we will see shortly, by adding all those Δy values over a range of input (which is represented by the Sigma or "summation" operator in the Visual Model), we can recover the original output y.

[4] More precisely, we need to indicate we are integrating with respect to variable x, but we omit it here for brevity.

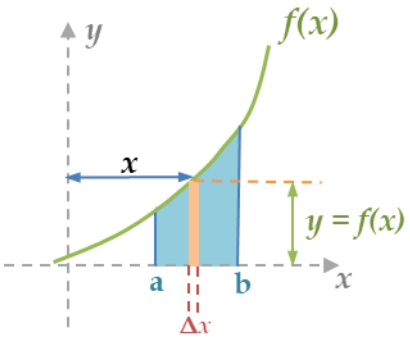
If we know the change in output (Δy) at a given input x, how do we calculate y value (the original output of $f(x)$) at x? To do that you need to *add* Δy values sequentially starting from a known input and output pair (say x_0 and y_0) as shown on the right. This figure shows that if we know the value of y_0 (at input x_0) then we can add Δy_1, and Δy_2 to get y. We can do this summation repeatedly to find the required y value. That's why integration is often viewed as a **summation**. In fact, the integral sign is an elongated "S", indicating summation. The right Visual Model above, shows this summation of all Δy values to produce y.



The following figure summarizes the relationship between differentiation and integration. The top figure shows how we can calculate the derivative for $f(x)$ at input x – i.e., we find Δy for given Δx and take the ratio. If we plot the values of those ratios, we get the graph at the bottom, which is $f'(x)$. Therefore, on the bottom graph, for the same input value x, we get output value $\Delta y/\Delta x$ (because this graph represents the derivative of the top curve). Therefore, the area of the orange rectangle gives us $\Delta y/\Delta x \times \Delta x$, which evaluates to Δy (area of a rectangle is its height multiplied by its width). Once we know Δy, how do we calculate y? As we saw above, we need to add Δy values starting from a known position. For this example, we see that $f(0) = 0$. Therefore, we can start the process at x=0 and sum all of the resulting orange rectangles to get the output y value of function $f(x)$. This sum is represented by the green area because, if we keep on adding all rectangles under the $f'(x)$ curve from x=0 to x, we get the green area. This also shows how integration can be used to find the area under a curve. We do that by repeatedly adding rectangles like the orange rectangle (Δy areas) shown above. This is the reason why integration represents a summation (sum of Δy rectangles) and often used in finding area of surfaces (and volumes of solids).



The computer code for calculating the numerical integral of a given function over a given input range (a to b) is given below. The given algorithm calculates the integral by finding the area under a curve, as shown on the right. There, we add up the area of each orange rectangle from input values a to b, to calculate the area of the blue region. Similarly, the function `getIntegral` accepts a function `func`, along with the start and end value of the input range (`a` and `b`). First, we initialize the running sum (`area_sum`) to zero to indicate that we have not calculated any area under the curve yet. Then, we change `x` from the start of the input range (a) to end of the input range (b) with a step size of `deltaX`. For each of those input `x` values, we calculate the output `y` value, and then calculate the area of the narrow rectangle (the orange rectangle above) with height `y` and width `deltaX`. Then, we add that area of rectangle to the running sum (`area_sum`). After we are done with the loop, we return `area_sum`, which is the total area under the curve of function '`func`', for the input range from `a` to `b`. If we wanted to graph the integral (e.g., in the case of a scientific calculator) we could do so by printing pair (`x`, `area_sum`) in the loop.




```
function getIntegral( func, a, b )          // get integral from a to b
{
      deltaX = 0.0001                       // small delta value, Δx
      area_sum = 0.0                        // area under the curve

      for x=a to b, deltaX                  // loop a to b with step Δx
      {
            y = func(x)                     // get output y for inp x
            area = y * deltaX               // area of small rectangle
            area_sum = area_sum + area      // add to total area
      }

      return area_sum                       // return area under curve
}
```

```
function myFunc( x )                        // function to integrate
{
      return 3*x*x + 2x + 1                 // 3x²  + 2x + 1
}
```

```
y = getIntegral(myFunc, 0, 5)               // y = integral(myFunc,0,5)
print( y )                                  // print integral
```

Summary

Since this is not a book on calculus, we are not going to go deep and learn how to calculate the derivative and integral of different functions. When you study Calculus, you should remember to organize the derivatives and integrals using the function families we learned in Chapter 4 (i.e., note the derivative and integral of each function family, its inverse and reciprocal).

I hope this short introduction helps you to organize your thoughts about how to think about calculus – just as another set of useful operators that help us build models about the real world. Once you look at differentiation and integration as applying operators (i.e., functions) and treat them as inverse functions of each other, all the intuition we built on functions can be readily applied to them.

ACKNOWLEDGEMENTS

First, I want to express my sincere gratitude to my wife Nirasha for supporting me throughout the long process of writing this book. I never intended to write such a long book, and it would not have been possible without the backing of my family. Additionally, hats off to many of my friends, who provided valuable feedback on the early versions of this book. Proofreading a math book of this size is no simple task, and thus, I must extend my gratitude to my son, Nisala Kalupahana, for undertaking that challenging task.

Ruchira Sasanka

INDEX

A

abstraction, 60, 179
algebraic function, 148
algebraic vector, 205, 220
algebraic vectors, 197
amplitude, 124
Amplitude Modulation, 123
angular frequency, 121, 167
arccos, 128
arcsine, 128
area under a curve, 261
array, 141

B

base, 94
basis vectors, 196
bell curve, 132
bias, 85
binary number, 143
Boyle's Law, 111

C

calculus, 257
capacitor, 167, 169
carrier wave, 124
Cartesian basis, 196
Celsius, 84
Coefficient Matrix. *See* Matrix Multiplier
column view of a matrix, 237, 241
complex number, 157, 246
complex plane, 157
complex roots, 162
component representation, 197
conjugate, 158, 162
constant term, 82, 85
convex lens, 62
cosecant, 120, 129, 136
cosine, 119, 147, 148
Coulomb's Law, 113
cross product, 192
cross product., 190
cube root, 102
cubic model, 90
current, 73

D

Decibels, 108
decimal number, 143
dependent variable, 16
derivative, 145, 258, 260
differential operator, 258
differentiation, 261
directly proportional, 75
domain, 24
dot product, 186, 198, 223, 247

E

electrical power, 79
equation, 16, 40, 41, 47, 51, 53, 54
Euler's Identity, 172
evaluation, 22
even symmetry, 80
exp, 109, 147
exponential decay, 115, 136
exponential growth, 97
exponential model, 94, 105, 131, 136
exponentiation, 66
expression, 19, 22

F

factor, 92
factorial, 129, 136, 147
factors, 88
Fahrenheit, 84
Ferris wheel, 117, 169, 174
for loop, 139, 140
force field, 209
Fourier series, 149, 167
Fourier Transform, 51, 174, 176
frequency, 117, 120, 167
Frequency Modulation, 125
function, 13
function composition, 27, 28, 31, 32, 33, 43, 44, 107, 128, 133, 135, 232, 234, 239
function decomposition, 32
function definition, 17
function evaluation, 21
Fundamental Theorem of Algebra, 91, 161

G

Gaussian Distribution, 132
Gaussian Elimination, 243
geometric vectors, 197
gradient, 209, 258
gradient field, 209
graph, 23, 163
Gravitational Law, 113

H

half-life, 116
hyperbolic function, 136

I

Ideal Gas Law, 76, 111
identify function, 112
identity function, 73, 100, 101
identity matrix, 238
identity transformation, 238
if-else statement, 95
imaginary number, 155
impedance, 167, 168
independent variable, 16
inductor, 167, 168
infinite series, 142
integral operator, 260
integration, 261
inverse, 42, 99, 260
inverse square model, 113
inversely proportional, 109
irrational number, 68, 144

L

Laplace Transform, 51, 176
linear combination, 86, 91, 122, 136, 195, 197, 200, 205, 218, 220, 223, 231, 242, 247, 249
linear model, 83, 103
linear term, 73, 110
linear transformation, 237, 248
linear-map, 248
logarithmic model, 105, 106, 136
logistic growth model, 135

M

Maclaurin series, 144
magnifier, 15, 26, 62, 73
mathematical relation, 12
matrix, 221, 246
  inverse, 243
matrix multiplication, 231, 232, 234, 238, 240
matrix multiplier, 225, 248
models, 71
moment, 190
multiplicative inverse, 110, 243

N

natural exponential function, 98
natural logarithm, 106
Normal Distribution, 132

O

odd symmetry, 82
Ohm's Law, 73
operators, 26

P

parameter, 34, 213
period, 117
periodic functions, 117, 149
permutations, 130
phase shift, 120, 167
phasor, 168
polar representation, 173
polynomial, 82, 90, 91, 104, 114, 138, 161
position vector, 208
power function, 71, 131, 136
power series, 143
pressure, 77, 111

Q

quadratic formula, 104
quadratic function, 87
quadratic term, 78

R

radians, 120
radioactive decay, 115
range, 24
rate of change, 257
rational numbers, 61
real number, 68, 246
reciprocal relationship, 109
recursion, 96, 130, 131
resistance, 73, 111, 166
Richter scale, 108
root, 52, 54, 89
row view of a matrix, 237

S

scalar, 179
scalar coefficient. *See* scalar multiplier
scalar field, 207, 208, 251
scalar multiplier, 200, 225, 247, 248
scalar product. *See* dot product
scientific notation, 66
secant, 129, 136
second law of motion, 75, 185
sense, 58
series, 90
Sigma notation, 138, 142
sign, 58
signal, 128
sine, 118, 147
slope, 258
solutions, 40, 53
square root, 67, 101
standard basis, 196
summation, 261
symmetry, 86
system of linear equations, 242

T

tan, 120
Taylor series, 149
temperature, 77, 206
Tensor, 246
term, 19, 22, 72
torque, 190
transcendental function, 148
transformation matrix, 238
trigonometric model, 123, 136

U

universal gas constant, 76

V

variable, 16
vector, 179, 220, 246
vector coefficient, 225
vector field, 209, 213, 251
vector multiplier, 201, 225, 248
vector product. *See* cross product
vector valued function, 213, 214
Visual Model, 14
voltage, 73, 169
volume, 77

W

waveform, 126, 149
whole number, 56, 142, 143

Z

zeros, 52

About the Author

Ruchira Sasanka is a computer engineer working in the field of High Performance Computing trying to eke out every bit of performance from large-scale scientific applications running on supercomputers. He received his M.S. and Ph.D. in Computer Science from University of Illinois at Urbana-Champaign, B.Sc. in Computer Science & Engineering from University of Moratuwa, Sri Lanka, and holds many US patents. When not grappling with computers, he spends his time hiking, camping, and landscaping. He lives in Oregon, with his wife, son, and his imaginary dog.