Advanced Math and Functional Programming, 2022 Mississippi Governor's School

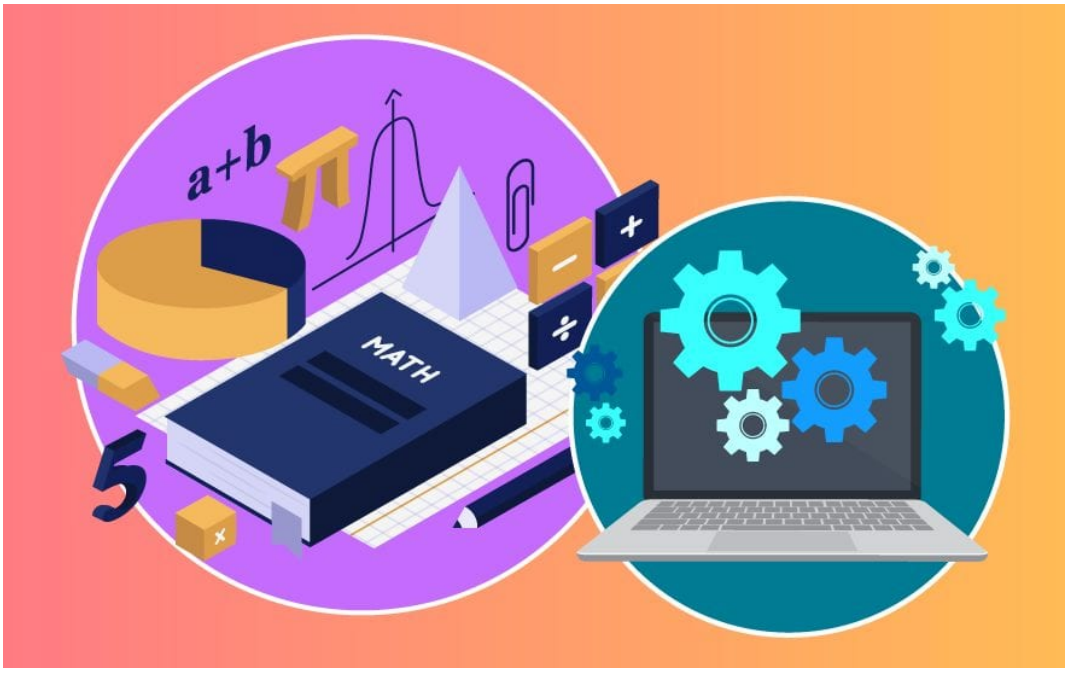
Dr. Jim Newton

June 13, 2022

Contents

Chapter 0

Introduction



0.1 Objectives

There are several objectives of this course:

Mathematics: Ameliorate your love for Mathematics.

Computer Science: Develop a sense of functional programming.

Communication: Confidently defend your ideas.

Why math? We assume that you as a student already loves math, and we will use examples in abstract math as source of ideas to write computer programs.

The mathematical and computer science topics have been chosen for this course complement each other. You will enforce what you've learned by writing computer programs using the ideas and also by explaining to your peers what it is that you have done.

It is important that the leaders of tomorrow, especially the leaders in the technology sector, understand the impact on humans of technological development. We must know how to apply abstract concepts to real computation problems, and how to explain their ideas and defend their decisions to their peers.

Don't forget that you are working not only with machines but also with humans. In this course, we will maintain a respectful, inclusive, and safe environment. Remember that different people have different strengths and weaknesses. Scientific discussions can sometimes become heated and experts might disagree about the best course of action. We will keep disagreements objective and use science to test ideas, rather than relying on emotions.

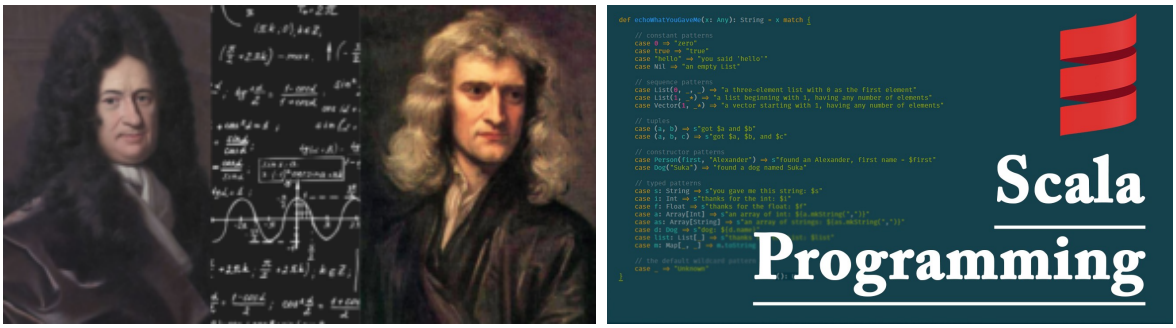
Good ideas that are never explained die and are forgotten. It is my belief that you don't understand something until you can explain it. From time to time, some students will present their solutions to their peers. This means defending your choices, recognizing strengths and shortcomings, and being subjected to peer review (important in the scientific process). This exercise is intended not only to learn to accept constructive criticism but also to build confidence.

0.2 Note from your Instructor

I'm Dr. Jim Newton and I will be your instructor and coach for two weeks as explore mathematical applications in functional programming.

I am a Mississippi native, and a graduate of Mississippi State University where in 1988 I received a BS in Electrical Engineering and in 1992 an MA in Mathematics. Years later, in 2015, I received my PhD in Computer Science from Sorbonne University in Paris, France. I currently live in Issy les Moulineaux, France, and work as an assistant professor and researcher at EPITA (`www.epita.fr`), a French engineering school

In 1981, I was a scholar at the first ever Mississippi Governor's School. I've designed this course remembering my own experience at MGS, back in the day. I realize that different students coming from different high schools in Mississippi have been exposed to various levels of mathematics and computer science courses. I nevertheless hope that regardless of your background and previous experience, you will find this course interesting and challenging. I am confident that if you apply yourself and remain self-confident, you will succeed and learn a lot.



0.3 Overview

We will probably proceed through up to 5 units. The first unit is special, dealing with setting up the environment. Units 2 through 5 are each organized in two stages:

- Theory — Learn mathematics and computer science concepts.
- Practice — Create code relating to the theory of various units.

There are two remaining sections (6 and 7) which you can continue with after MGS if you find them interesting and challenging.

We will primarily cover introductions to two mathematical areas: Abstract Algebra and Analysis (the fundamental principles of Calculus). Our practical treatment of Algebra and Analysis will help you prepare for a more theoretical treatment at the university level.

We will use a functional programming language called Scala, `https://www.scala-lang.org`. Functional languages are considered by some to be more difficult than imperative languages, and students rarely start programming using a functional language. Even if seen by many people as being exotic, you will find that a functional programming language can be a natural way to elegantly express certain mathematical concepts. You will later be able to take the things you've learned and apply them back to other (more popular) programming languages such as Python, Java Script, Lisp, C++, etc.

0.4 Proposed Syllabus

Week	Day	Unit	Activities
1	Mon	0, 1	Environment Set-up, Hello World
1	Tue	2	Sets and Functions, Theory, First Program
1	Wed	2	Coding Exercises, Presentation
1	Thu	3	Abstract Algebra, Theory
1	Fri	3	Group Project, Coding Exercises
2	Mon	3	Review, Coding Exercises
2	Tue	4,5	Analysis: Convergence, and Sums
2	Wed	4,5	Coding Exercises, Presentation
2	Thu	6,7	Analysis: Derivative, Integral
2	Fri	6,7	Coding Exercises, Presentation

Chapter 1

Development Flow

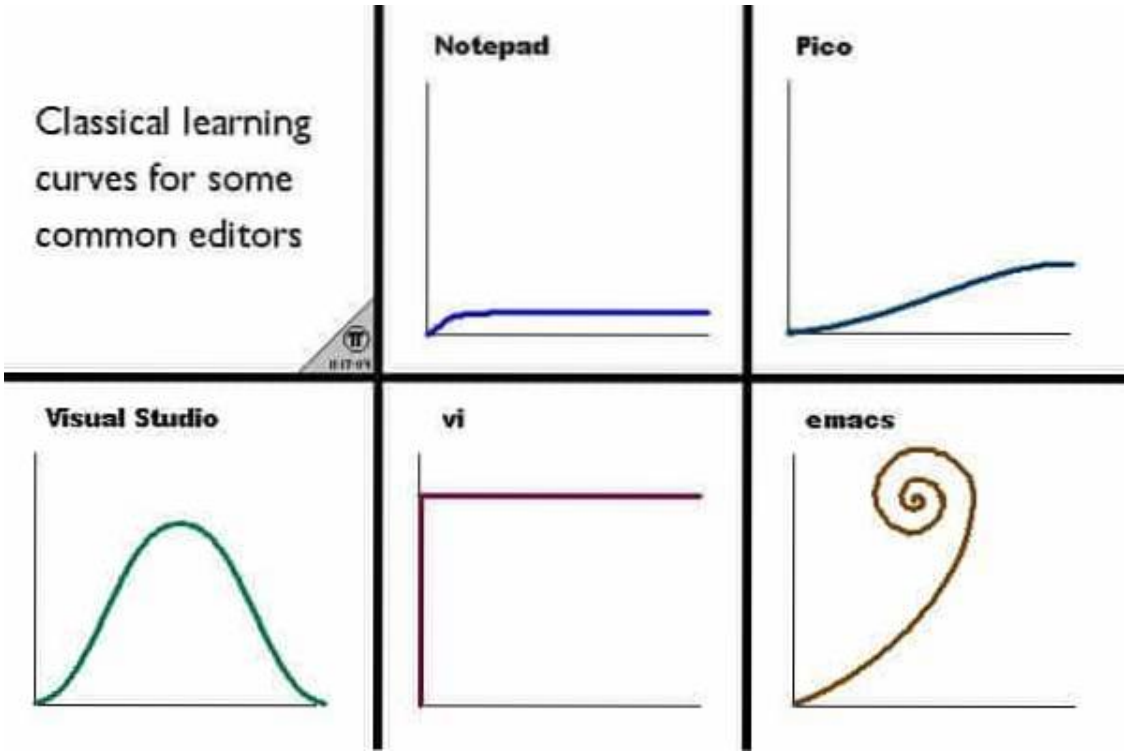


Figure 1.1: Some editors are easy to learn, others are very powerful after you learn them.

1.1 Editors

To develop a computer program you must create and edit text files. These are files containing only characters. This *can* be done using a simple text editor such as Notepad, or a more powerful editor such as *emacs* or *vi*. Using an editor means the programmer must have a more complete understanding of the compiler and the build environment. Many expert programmers prefer this approach as it gives more low-level control to the user.

We will use an IDE (interactive development environment) rather than a text editor for this project.

1.2 VS Code

An IDE allows you to edit code as a text file. It also allows testing and debugging. Learning a new programming language (Scala in our code)

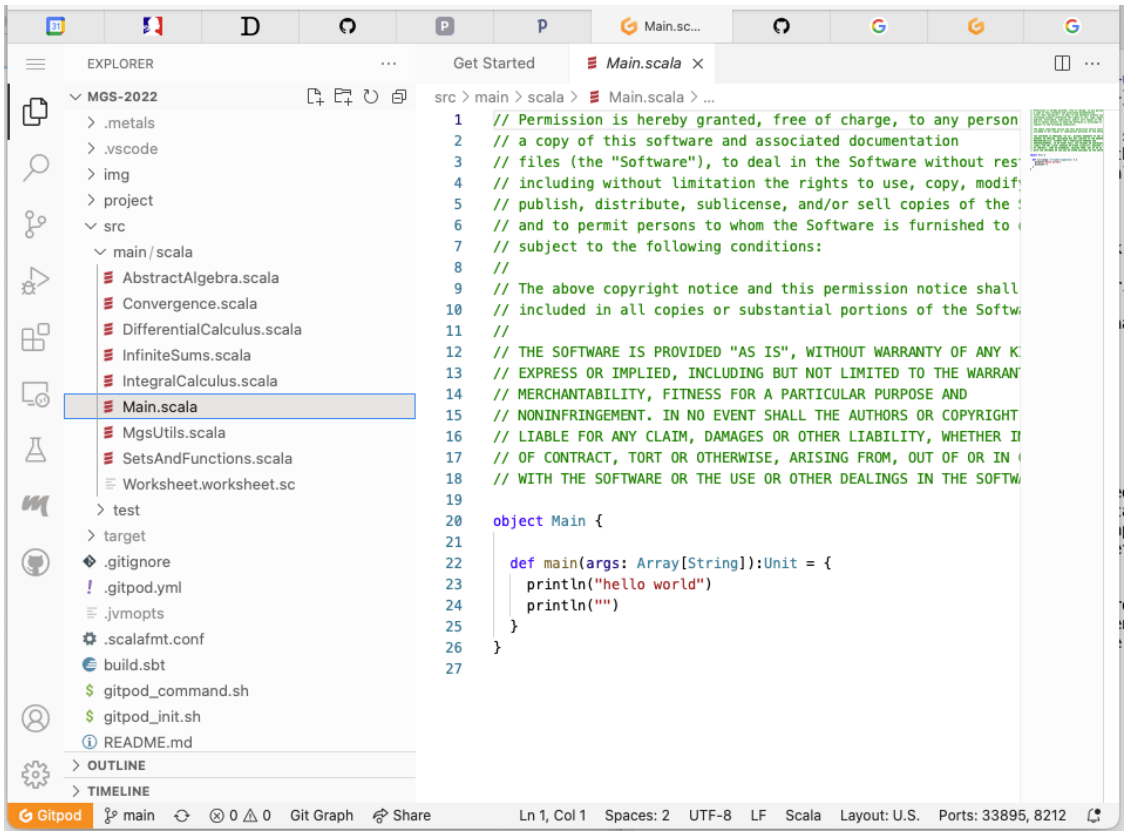


Figure 1.2: The VS Code IDE runs stand-alone, or in the GitPod development environment. The GitPod/VS Code integration allows you, the developer, to access remote a repository using only a web browser.

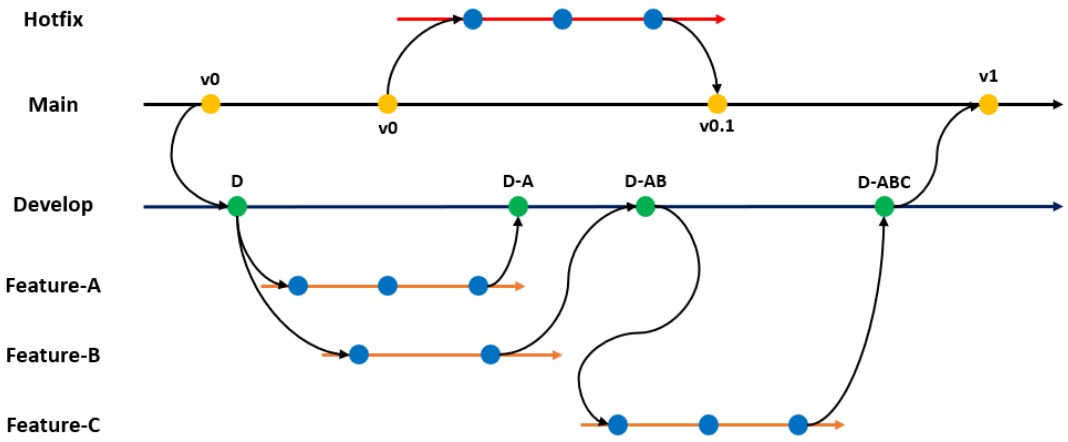


Figure 1.3: Version Control allows us to manage development points in out project.

is much easier with an IDE as the environment provides many development hints such has highlighting, error detection, coding suggestions, and completion.

An IDE often has a *plug-in* for the programming language you are using, because syntax, naming, and usemodel may be very different from language to language.

The most popular IDE today for working with Scala is probably IntelliJ. However, setting up Scala on IntelliJ is often time consuming. Instead, we will use *VS Code*[1] which comes pre-installed in the cloud environment which we will use. VS Code is a free and open source product made by Microsoft for Windows, Linux, and MacOS.

1.3 Version Control

A version control system allows us to maintain a history of our work; see Figure 1.3. We can revisit previous states of our project. We can also create branches to develop different features in parallel without them interfering with each other.

An important feature of modern version control systems is collaboration. Developers can easily share their work with other developers.

[1] `https://code.visualstudio.com` and `https://en.wikipedia.org/wiki/Visual_Studio_Code`

We will use *git* as the version control system. In recent years git has become ubiquitous in the field of open source software development. There are many videos on YouTube to help a person learn git. We will interact with git exclusively through our IDE. The IDE hides most of the difficulty of performing simple tasks, but if something goes wrong, an IDE is not very good for debugging version control issues.

1.4 GitHub

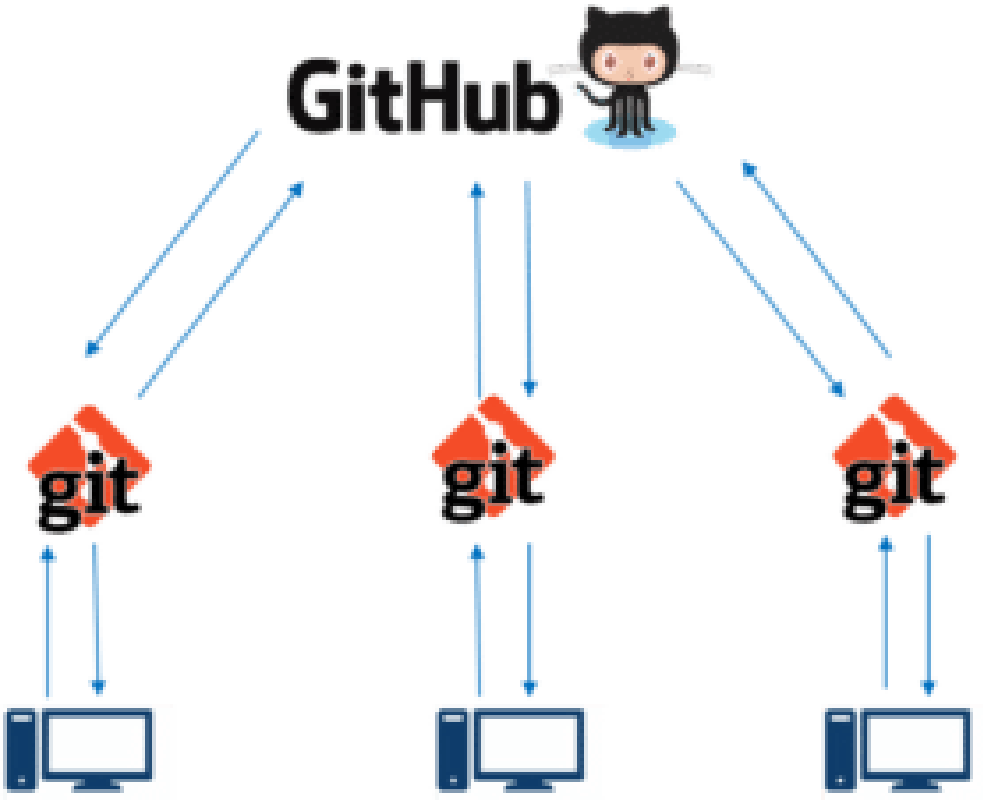


Figure 1.4: GitHub is a cloud service which hosts code repositories. You connect to GitHub either by web browser or directly from *git*.

In this course we will be developing software together. You will be able to share your code with your classmates, and with the world. However, you control when you want to make your code available to the

world, and when you want to incorporate your classmate's code into your development area.

To achieve this goal we have a central repository of code, hosted by GitHub on the cloud. You will download (clone) a copy of this repository, edit existing code, and create new code, and then occasionally upload your contributions back to the central repository, making your contributions available to everyone else.

GitHub is a web based service which hosts thousands of projects, small and large, in the cloud. Typically, users have a development environment on their local computer (laptop or desktop) and periodically upload and download changes to the cloud.

The URL `https://github.com/jimka2001/mgs-2022`, is the public entry point to the GitHub project which we will work on in this MGS-2022 course. How we will edit code in the GitHub project is described in Section 1.5.

You must create an account on GitHub (if you don't have one already). This will be done in Section 2.4.

1.5 GitPod

Most people install software such as IDEs (integrated development environments), editors, compiler, language implementations, version control software, onto their personal computer or laptop. You are free to do that when you return home. But for the entirety of MGS-2022, we'll use a development environment entirely on the cloud.

We will use a web based service called, GitPod to provide an environment for working with GitHub. This environment and all the code you develop will still be available to you after MGS-2022 is finished.

The GitPod web site is `https://www.gitpod.io`.

Figure 1.2 shows an image of what the development environment looks like.

GitPod provides a version control interface. You should be able to do all code editing, and collaboration efforts (push, pull, view history) using this interface.

Figure 1.5: GitPod provides a web-brower-based development environment which integrates VS Code with GitHub.

1.6 Setting up the environment

It may be a bit tedious to set up the development environment, because before you can learn to use a piece of software, you have to install it; necessarily you are installing things you don't yet understand. The following steps should help, but often things go wrong with set-up, so we'll troubleshoot as we go.

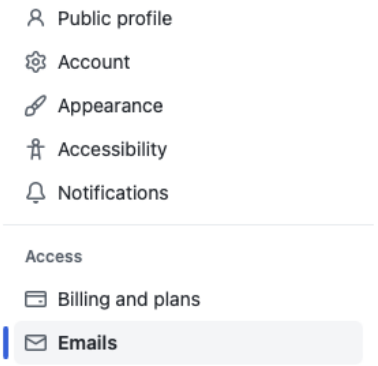
1. Create a GitHub account using an abstract user name. Don't use your real name. This URL, `https://git-scm.com/book/en/v2/GitHub-Account-Setup-and-Configuration` explains how to create a GitHub account. There are many similar web-sites, e.g., `https://www.wikihow.com/Create-an-Account-on-GitHub`. Primarily this is self explanatory at the URL `https://github.com/join`.

2. (Optional) Configure your GitHub account with an profile image. Don't use a picture of yourself, rather use an abstract avatar such as `https://www.flaticon.com/search?word=animal+avatar`.

3. Configure GitHub to keep your email address private.

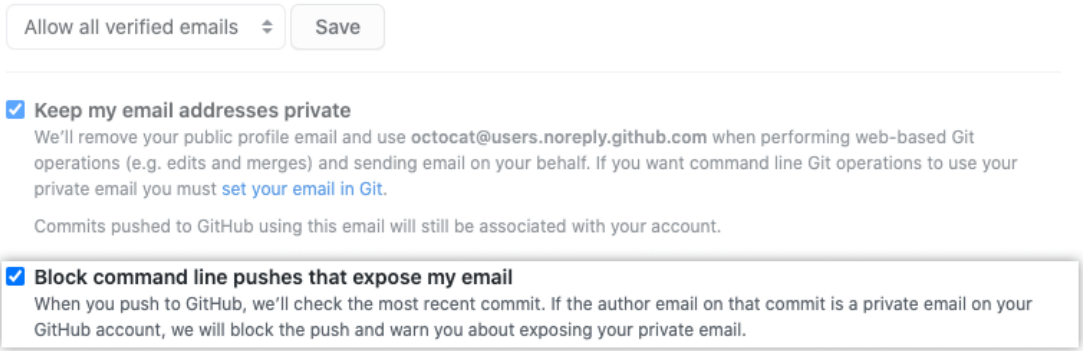
(a) In the upper-right corner of any page, click your profile photo, then click **Settings**.

(b) In the "Access" section of the sidebar, click **Emails**.



(c) To keep your email address private when performing web-based Git operations, click **Keep my email addresses private**.

(d) To keep your email address private in commits you push from the command line, select **Block command line pushes that expose my email**.



4. Open `https://github.com/jimka2001/mgs-2022` with your web browser.

5. Fork the repository. See Figure 1.6.

6. Open the web-ide with GitPod, you may just prepend `http://gitpod.io/#` to the URL already in the web browser.

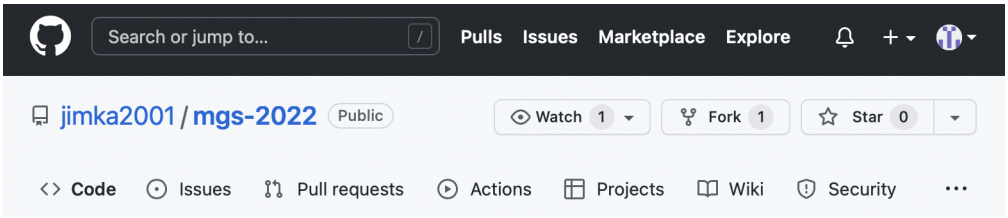


Figure 1.6: GitHub Fork

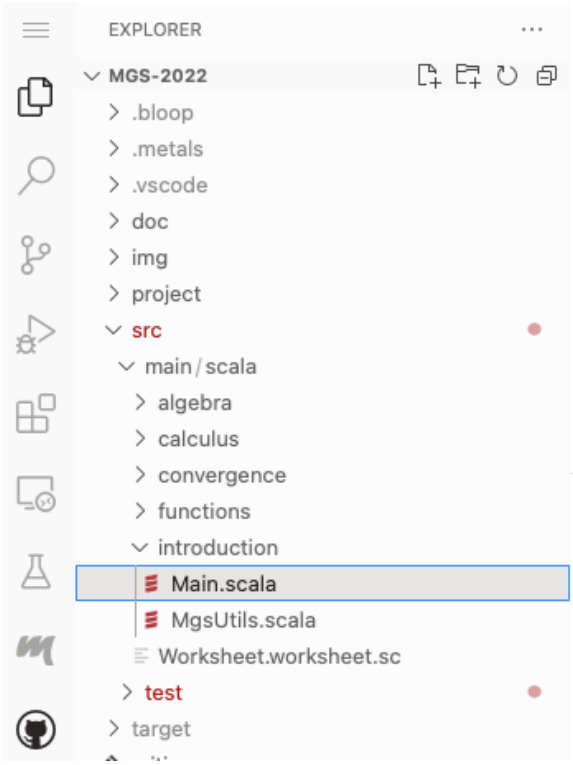


Figure 1.7: Opening `Main.scala`

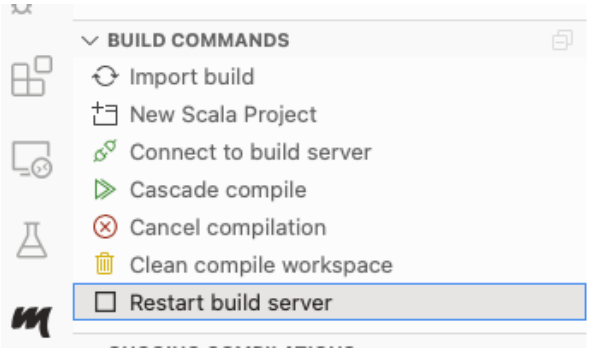
Figure 1.8: The editor pane of the `Main` object.

7. Find the Explorer, and open it to find `Main.scala` at `src/main/scala/introduction/Main.scala`. See Figure 1.7.

8. Click `Main.scala` to open the file in an editor pane. You should see something like that shown in Figure 1.8.

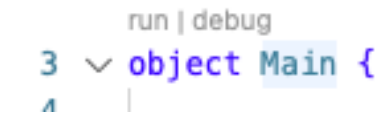
9. The `run | debug` link between lines 2 and 3 may not appear immediately, you may have to wait a couple of minutes. If you don't see **run debug**, you may need to restart the build server, and recompile. Find and click the Metals icon  . This should open the Metals panel where you can **Restart the build Server** and **Cascade compile**.

Figure 1.9: The editor pane after running the `Main` object.



Thereafter, you should be able to find the **run**—**debug** link in the Scala code.

10. Find the definition `object Main`. If you click on `Main` you should see something like this:

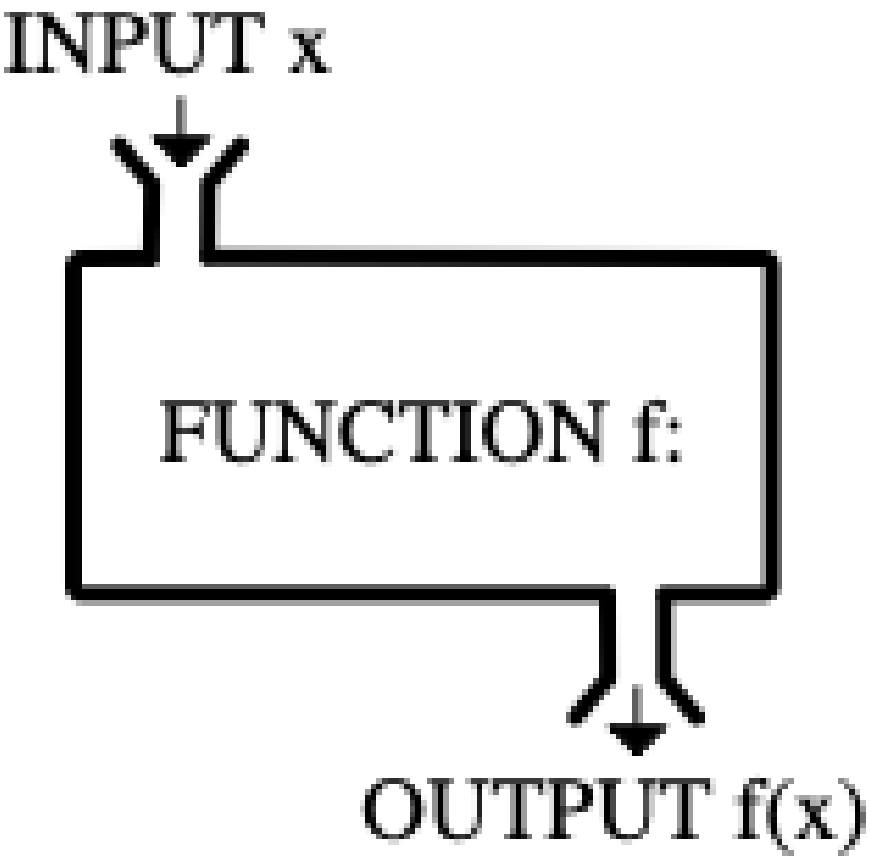


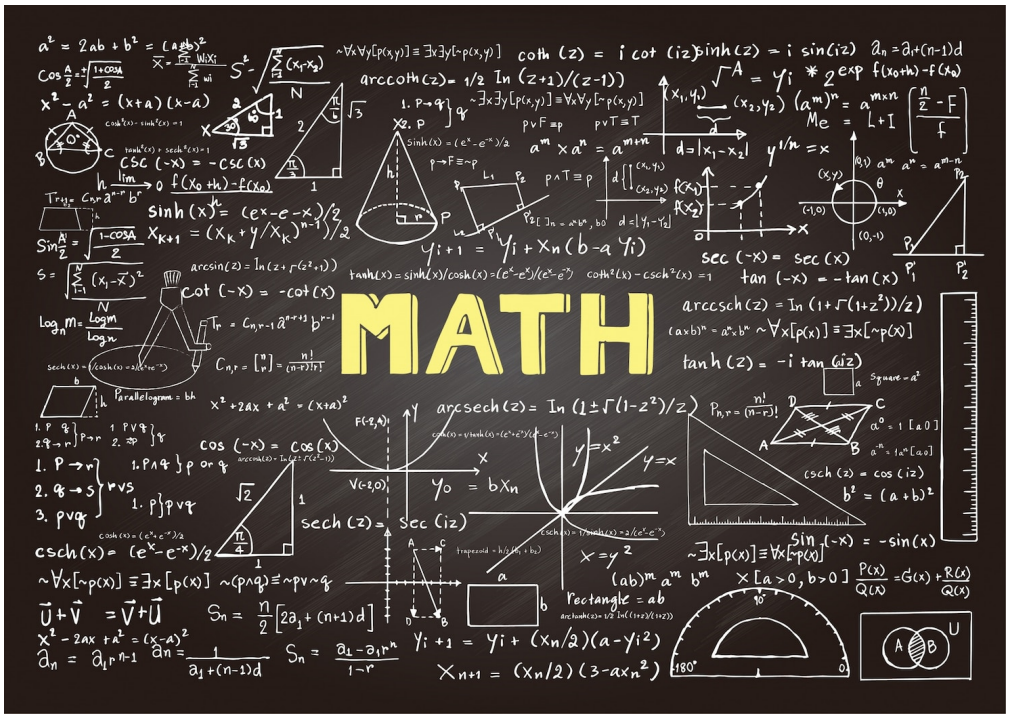
Click **run**.

11. You should see a new window pane appear below the program text showing `hello world`, as shown in Figure 1.9.

Chapter 2

Sets and Functions





In this unit we will look at functions both from a mathematical perspective and also from a programming perspective. There are some important similarities and some important differences.

2.1 Mathematical perspective

We would like to talk about functions. But where should we start the conversation?

You may already have some intuition about functions. Sometimes functions have names, like sin, cos, and log. Sometimes functions lack names but have formulas such as $\frac{x^2-1}{x^2+2x+1}$.

2.1.1 Specifying functions

What is a function? A function such as

$$f : X \to Y$$

is a correspondence between two sets. But it is a special correspondence. In particular the function, f, associates a unique, well-determined, ele-

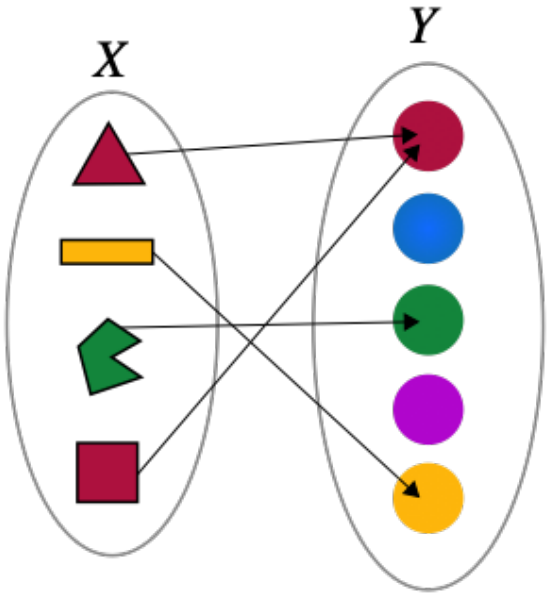


Figure 2.1: A function is a well-defined correspondence of all the elements of one set (the domain) with some of the elements of another set (the range).

ment of Y with each element of X. More precisely, if $x \in X$, then $f(x)$ designates a unique, well-determined, element of Y.

$$x \in X \implies f(x) \in Y \, .$$

Conceptually, a function has three parts

1. A *domain*, i.e., a place from which input values may be taken.
2. A *range*, i.e., places where output values are found.
3. A *rule*, i.e., a way of determining the output value given only the input value.

To implement a function in the Scala programming language we have to specify the *domain* and *range* in terms of what are called *types*, and we designate the *rule* in terms of a procedure.

2.1.2 Mathematical notation

$$f(x) = 3x + 1 \tag{2.1}$$

Let's start with a very simple function (2.1). Actually, the only thing specified here is the rule. This function is written in a mathematical notation where the domain and range are not specified. The same notation can be used for different choices of domain and range. For example, this function works for integers, $\mathbb{Z}$, as input, but it will also work for natural numbers, $\mathbb{N}$, rational numbers, $\mathbb{Q}$, real numbers, $\mathbb{R}$, complex numbers, $\mathbb{C}$. The same function will also work for matrices or for functions themselves.

If we want to be more precise, we can use a more descriptive notation as in (2.2)

$$f : \mathbb{N} \to \mathbb{N} \text{ by } f(x) = 3x + 1 \tag{2.2}$$

The domain and range of a function need not be the same, and need not be a set of simple numbers. For example, we use the symbol $\mathbb{R}^2$ to denote the set of ordered pairs of real numbers; i.e.,

$$\mathbb{R}^2 = \{(x, y) \mid x \in \mathbb{R} \wedge y \in \mathbb{R}\}.$$

Equation (2.3) shows an example of such a function.

$$f : \mathbb{R}^2 \to \mathbb{R} \text{ by } f(x, y) = 3x - 2y + 1 \tag{2.3}$$

Mathematical notation such as $a + b$, $a \times b$, $a - b$, a^3, and $\frac{x^2-1}{x^2+2x+1}$, but we may also specify functions using cases, like in Equation (2.4)

$$|x| = \begin{cases} x & ;\text{if } x > 0 \\ 0 & ;\text{if } x = 0 \\ -x & ;\text{if } x < 0 \end{cases} \tag{2.4}$$

2.1.3 Specifying functions by recurrence

Functions can be defined in terms of themselves. We might use an intuitive definition such as Equation (2.5).

$$x^n = \underbrace{x \times x \times \ldots \times x}_{n \text{ times}} . \tag{2.5}$$

We can define the same function more explicitly by using recursion as in Equation (2.6).

$$x^n = \begin{cases} 1 & ; \text{if } n = 0 \\ x \times x^{n-1} & ; \text{if } n > 0 \\ \frac{1}{x^{-n}} & ; \text{if } n < 0 \text{ and } m \neq 0 \end{cases} \tag{2.6}$$

Similarly, we can define the factorial as

$$n! = 1 \times 2 \times \ldots \times n$$

or we can define it as

$$n! = \begin{cases} 1 & ; \text{if } n = 0 \\ n \times (n-1)! & ; \text{if } n > 0 \end{cases} \tag{2.7}$$

The first case need not be $n = 0$. We can define the Fibonacci numbers. Let $F(n)$ denote the n'th Fibonacci number.

$$F(n) = \begin{cases} 1 & ; \text{if } n = 1 \\ 1 & ; \text{if } n = 2 \\ F(n-1) + F(n-2) & ; \text{if } n > 2 \end{cases} \tag{2.8}$$

Likewise, we can define the set of subsets of size n. We use set comprehension notation, see Section 2.2.

$$\mathbb{P}_n(S) = \begin{cases} \{\emptyset\} & ; \text{if } n = 0 \\ \big\{\{x\} \cup y \mid x \in S, y \in \mathbb{P}_{n-1}(S \setminus \{x\})\big\} & ; \text{if } n > 0 \end{cases} \tag{2.9}$$

Notice that Equation (2.9) specifies a function, not of numbers, but of sets. Its input is a set, and its output is a set of sets. I.e., the domain of $\mathbb{P}_n$ is a set of sets, and its range is a set of sets of sets.

2.2 Set comprehension

One useful way to define subsets is with a set comprehension. In its simplest form, it specifies the subset of elements which obey a given predicate. A predicate is a Boolean valued function, *i.e.* a function whose value is either $true$ or $false$. If S is a set, and the p is a predicate defined on S, then the subset of elements, x, for which $p(x)$ is $true$ is denoted:

$$\{x \mid x \in S, p(x)\}.$$

We may also compose conjoin multiple prediates such as

$$\{x \mid x \in S, p(x), q(x)\}.$$

For example, the set of even integers between 0 and 100 inclusive may be denoted as such:

$$\{x \mid x \in \mathbb{Z}, even(x), 0 \leq x \leq 100\}.$$

Finally, the value on the left hand side need not be a simple variable such as x; rather it may an expression. For example the squares of the even integers between 0 and 100 can be expressed as follows:

$$\{x^2 \mid x \in \mathbb{Z}, even(x), 0 \leq x \leq 100\}.$$

In Section 2.1.3, we saw a set comprehension which defines the subsets of size n. As a final example suppose $S = \{0, 1, 2, 3\}$, and that T is the set of all 2-element subsets of S which do not contain 2.

The set of 3-element subsets of S which contain 2 can be written:

$$\{\{2\} \cup y \mid y \in T\}.$$

2.3 Programming perspective

2.3.1 Functional Programming Languages

We will not spend much time talking about programming languages which are not *functional* but very briefly: imperative programming languages tend to model the machine, allowing you to phrase your programs in a way which the machine can interpret. Functional languages, tend to make the machine understand mathematical notations, which is a more natural and elegant way of thinking. Functional languages are often considered more difficult and more exotic. However, not everyone (including your instructor) agrees that they need to be considered difficult.

Imperative languages allow you to detail how a computation should be carried out. Functional languages allow you describe the problem in terms of what the solution looks like.

The web site and lots of documentation about the Scala language can be found here: `https://www.scala-lang.org`. The newest Scala version is 3.1.x; however, we will use 2.13.8 (or higher).

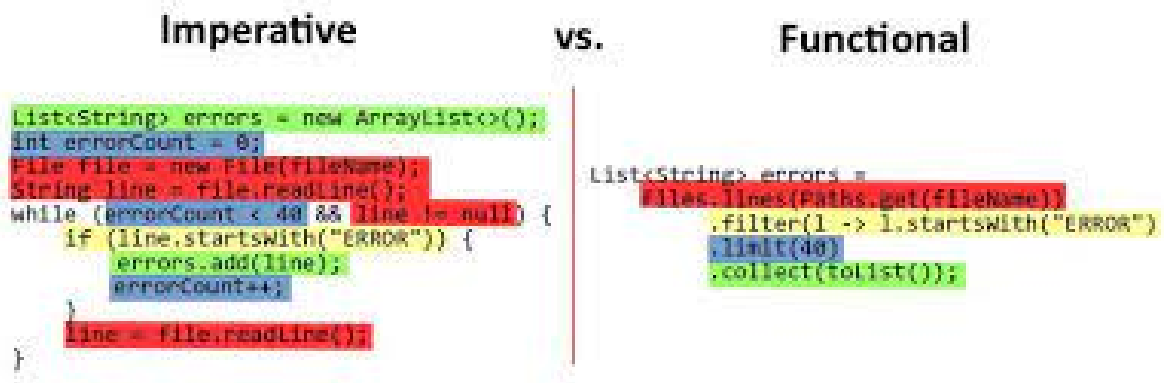


Figure 2.2: Imperative vs Functional programming

2.3.2 Elements of a Scala program

Figure 2.3 shows some of the elements of a Scala program.

1. Lines 7 through 15 define a function named `quadraticFormula`. This function has 3 arguments, `a`, `b`, and `c`, each of type `Int`—meaning that whenever the function is called, the call-site must provide 3 integer arguments.

2. Lines 12 and 13 are two mathematical computations computing the roots of a specified polynomial. These values are each double-precision floating-point values (each a `Double`).

3. Computing the values on lines 12 and 13 involves computing a square root, $\sqrt{b^2 - 4ac}$. The `sqrt` function in not normally visible to the programmer; it must be *imported*. The statement on line 3 imports the `sqrt` function name into the visible name space.

4. If $b^2 = 4ac$, then $\sqrt{b^2 - 4ac} = 0$. In this case the two values computed on Lines 12 and 13 will be the same (a root of order two), we specify that duplicate elements are removed from the list by `List(...).distinct` on lines 11 and 14.

5. The function, `quadraticFormula`, returns a value which has type `List[Double]`, which is a list of double-precision (64-bit) floating point numbers. The valued within the function computed in the tail position is the return value from the function. In this case, the

```scala
package functions

import scala.math.sqrt

object QuadraticFormula {

  def quadraticFormula(a: Int,
                       b: Int,
                       c: Int
                      ): List[Double] = {
    List(
      (-b + sqrt(b * b - 4 * a * c)) / (2 * a),
      (-b - sqrt(b * b - 4 * a * c)) / (2 * a)
    ).distinct
  }

  def main(argv: Array[String]): Unit = {
    println("Hello this is main of QuadraticFormula")
    println(quadraticFormula(1, 2, -3))
    println(quadraticFormula(-1, 3, 7))
  }
}
```

Figure 2.3: Some Elements of a Scala Program, definition of an object containing a function.

expression in tail position is the one comprising lines 11 through 14: `List(...).distinct`. Usually the tail position is the final value (bottom-most) in the function. But in the case of an `if/else` the tail-position value will be which ever branch of the conditional is taken at run-time.

6. Many other types are also available as variables, and function return values. Some types which we will encounter are `Boolean`, `String`, `Tuple[]`, `Unit`, as well as many more.

7. The function named `main` which must be defined as shown in line 17, is regarded as the starting point of the program. In this case, `main`, makes two calls to `quadraticFormula` on lines 19 and 20.

8. Function parameters are grouped with parentheses, and separated by commas.

9. Function calls are grouped with curly-braces.

10. The functions are defined with an `object`, starting on line 5 and ending on line 22. This object has the name `QuadraticFormula` which is the same as the function but with different case, i.e., different capitalization. In the Scala language, names are considered *the same* if they have the same spelling and the same capitalization.

11. Line 1 defines a `package` named `functions`, which normally is the same as the directory name the file resides in. This correspondence of directory name and package name is conventional and not absolutely required.

2.3.3 Some other programming constructs

1. Conditionals

2. Recursive functions

3. Collection types: `List[]`, `Tuple[]`, `Vector[]`, `Set[]`.

2.3.4 Live coding

We implement the `fixedSizedSubsets` in live-coding. The program will be modeled after equation (2.9) in Section 2.1.3.

We write some tests. Some of which pass, and some of which fail. To compute the number of subsets of size k of a set of size n we need to compute $\binom{n}{k}$ using the `factorial` function, which you will develop.

You will also implement `countFixedSizedSubsets` as part of your programming exercises. This function is also needed to pass the tests.

2.4 Coding Exercises

2.4.1 Implement the Quadratic Formula

As a first program in Scala you will implement the quadratic formula. recall that if

$$ax^2 + bx + c = 0\,, a \neq 0$$

then

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}\,.$$

- Find the function `quadraticFormula`, implement a test for this function. The Scala function implements the mathematical function $x_{1,2}(a, b, c) = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$. The function computes the roots of the polynomial ax^2+bx+c, assuming $a > 0$, and assuming $b^2 \geq 4ac$. The test should go in the file `QuadraticFormula.scala` in directory `src/test/scala/functions`. There is already a function stub you can use.

- Extend the tests. Remember that to test values of type `Double` you may need check whether the absolute value is less than some threshold (ε) rather than testing for exact equivalence.

- Enhance `quadraticFormula`: Notice that the current implementation is inefficient, because it computes almost the same thing twice.

Also if you call the function with `a,b,c` for which there are no real solutions, the function will return `NaN,NaN`. What is `NaN`? If you don't know, look it up with an internet search. You should refactor the function to define a variable `discriminant` whose value is

$$\Delta = b^2 - 4ac\,.$$

Then there are three cases, if $\Delta < 0$, if $\Delta = 0$, and if $\Delta > 0$. Refactor the function so that if there are no real roots it returns an empty list; if there is a unique solution, return a list of one number; and if there are two distinct solutions, return a list of those two solutions **in increasing order**. I.e., if you return `(a,b)` assure that $a < b$. Also update the test case to check for $a < b$.

2.4.2 Recursive Functions

- File `Power.scala`
  - Implement `power` for `Int` and `Double`. In the case of `Double` you should also handle the case of a negative exponent.
  - Implement `String` concatenation using the power model.
  - Implement `List` concatenation using the power model.
- File `Factorial.scala`
  - Implement `factorial`. You will use `factorial` in Secion 2.4.4.

Discussion: In the `factorial` function, do we need a case for $n = 0$ and also $n = 1$? How does this relate to the principle of induction? Look up *Principle of Induction* if you need to review it.

2.4.3 Fibonacci numbers

- Find the file `Fibonacci.scala`
- Implement a recursive function such that given n, the function returns the nth Fibonacci number, F_n.

- What is the largest Fibonacci number you can compute in Scala?
- Search on the internet and find the Binet's formula for computing the nth Fibonacci number. Implement this function and compare the results to the recursive implementation.

2.4.4 Counting subsets

- Find the file `CountSubsets.scala`.
- In Section 2.3.4 we implemented (will implement) `fixedSizedSubsets` as live coding. You should `git pull` to update your workspace to reflect the work done in class.
- Implement `countFixedSizedSubsets` in the file `CountSubsets.scala`. There is a comment on the function stub. You may need to access the internet to find the correct formulas needed.
- Make sure all the tests pass in `SetsAndFunctionsTestSuite.scala`.

Chapter 3

Abstract Algebra

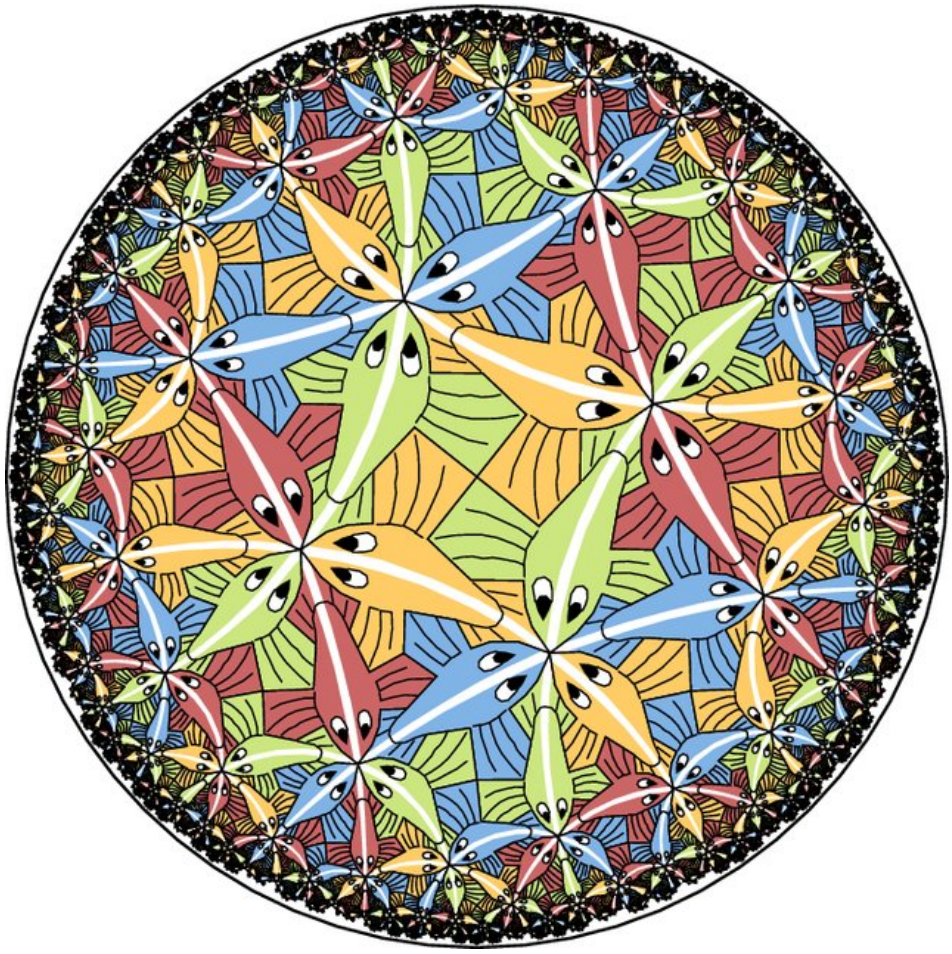


Figure 3.1: Evariste Galois died at 20 years old as a result of a duel.

In Section 2.4.1, you worked with a function which computes the (real) roots of a second-degree (quadratic) polynomial given only its coefficients. There is an analogous formula to compute the roots of a cubic (degree-3) polynomial, and still another to compute the roots of a degree-4 polynomial. Is there such formula for a polynomial of degree 5 (quintic) or higher?

The question of whether such a formula exists for the quintic had been an open question for 250 years until Niels Henrik Abel proved its non-existence in 1823 at the age of 21.

Around the same time a young, 20 year old, political activist of post-revolutionary France, Evariste Galois, attempted to explain why some higher-order polynomials are *solvable in terms of radicals* and others are not. I use the word *attempted* not to imply that his explanation was wrong. To the contrary, his ideas were correct but nobody understood his treatment of the subject. An example of his writing is shown in Figure 3.2. One well-respected mathematician of the time, Joseph Fourier, took home a sizable work of Galois to digest it but unfortunately died unexpectedly, and the work was nearly lost.

The story of Evariste Galois who quickly wrote down the principles of abstract algebra before his death as a result of a duel, is depicted in sev-

Figure 3.2: Galois's confusing handwriting

eral YouTube videos: notably *Evariste Galois a documentary* `https://www.youtube.com/watch?v=J6dsanpnpt0` and *Galois Theory Explained Simply* `https://www.youtube.com/watch?v=Ct2fyigNgPY`, both produced by Math Visualized.

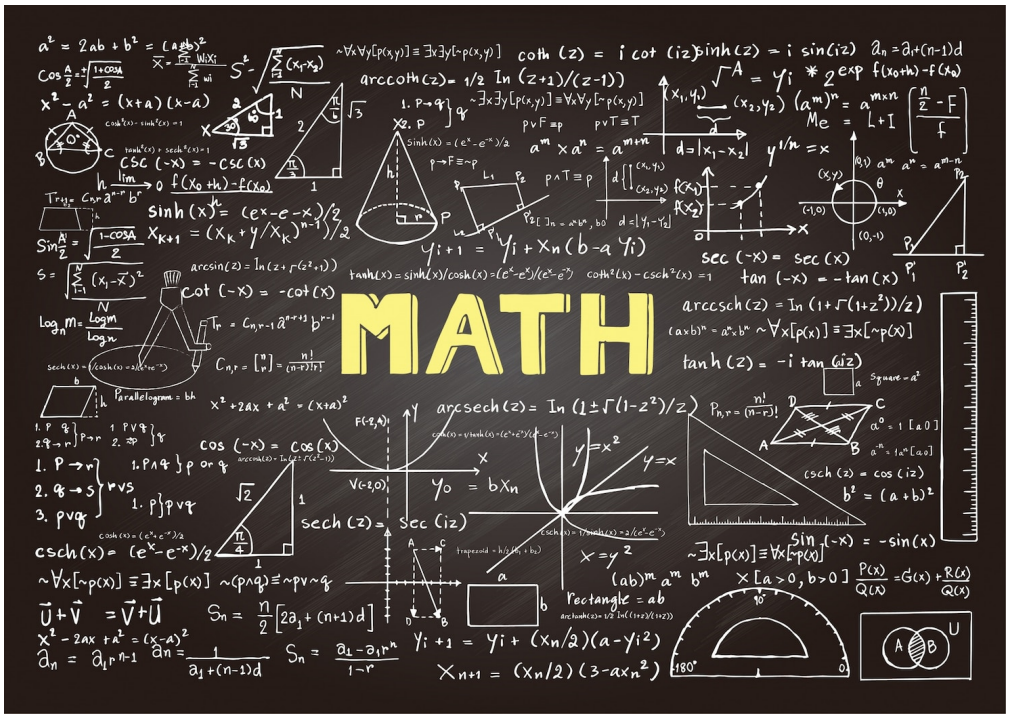
It is not surprising that mathematicians did not understand the work of Galois during his sort lifetime. Figure 3.2 shows an example of his handwriting.

In this unit we will study sets with mathematical structure. This approach to *abstract algebra* is of course much cleaner than that introduced by Galois. Galois somewhat flippantly introduced the *group* and

Figure 3.3: Galois, as a staunch republican, would have wanted to participate in the July Revolution of 1830 but was prevented by the director of the École Normale.

the *field* just as a tool to talk about the solvability of a polynomial, probably without realizing that they deserve in-depth study themselves.

There are several mathematical structures defined by axioms. We will thereafter write programs as individuals and as teams to manipulate these structures.



3.1 Mathematical perspective

3.1.1 Monoid

Definition 1 (Monoid)**.** $(S, \circ)$ *is called a* monoid *if*

1. *Closure:* $a, b \in S \implies a \circ b \in S$.
2. *Associative:* $a, b, c \in S \implies (a \circ b) \circ c = a \circ (b \circ c)$
3. *Identity:* $\exists e \in S$ *such that* $a \in S \implies a \circ e = e \circ a = a$

Notice that we assume that a monoid has an identity element, but we don't assume that it is unique. Can a monoid have more than one identity element: e_1 and e_2? The answer is no, but why?

Theorem 1. *The identity of a monoid is unique.*

Proof. Suppose e_1 and e_2 are both identities of monoid, M. Since e_1 is an identity, then $e_1 e_2 = e_2$. But since e_2 is an identity, then $e_1 e_2 = e_1$. Thus

$$e_1 = e_1 e_2 = e_2$$

□

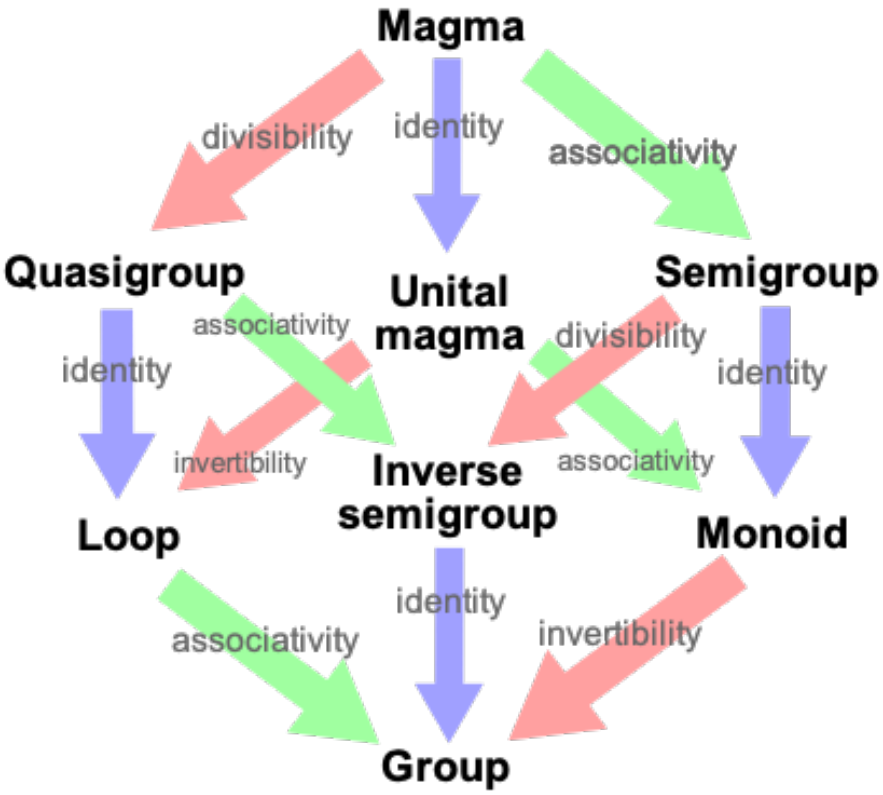


Figure 3.4: There are many algebraic structures which are linked by subset relations.

3.1.2 Examples of monoids

1. $(\mathbb{N}, +)$, the set of natural numbers under addition is a monoid.
2. $(\mathbb{Z}, -)$, the set of integers under subtraction is not a monoid. Why?
3. The set of even integers under addition is a monoid.
4. The set of even integers under multiplication is a not monoid. Why?
5. $(\mathbb{R}, \times)$, the set of positive real numbers under multiplication is a monoid.
6. The set of 2×3 matrices under addition is a monoid. (See Section 3.1.10 for a reminder of matrix addition.)
7. The set of 2×3 matrices under multiplication is not a monoid. However the set of 3×3 matrices under multiplication is a monoid. Why? (See Section 3.1.10 for a reminder of matrix multiplication.)
8. The set of subsets of a given set using the operation of union is a monoid. What is the identity element?
9. The set of subsets of a given set using the operation of intersection is a monoid. What is the identity element?
10. Let Σ be any set of distinguishable elements: $\Sigma = \{a, b, c, d\}$. Now consider the set, $\mathcal{L}(\Sigma)$, of all sequences of finite length $(x_1, x_2, \ldots, x_n)$ for which $x_i \in \Sigma$ for $i = 1, 2, \ldots, n$, $n \geq 0$. I.e., $\mathcal{L}(\Sigma)$ includes all sequences of length 12, and all sequences of length 54, and all sequences of length 10,051, etc.

    Let $+$ denote sequence concatenation. E.g.,

    $$(a, c, a, a) + (d, a, c, a, b) = (a, c, a, a, d, a, c, a, b)\,.$$

    $\mathcal{L}(\Sigma)$ is a monoid.

    What is its identity element?

    Is it a commutative monoid?

Figure 3.5: The movement of the hands of the clock is an example of a monoid arithmetic.

11. The integers on the face of the clock form a monoid under addition with the following addition table.

$+$	1	2	3	4	5	6	7	8	9	10	11	12
1	2	3	4	5	6	7	8	9	10	11	12	1
2	3	4	5	6	7	8	9	10	11	12	1	2
3	4	5	6	7	8	9	10	11	12	1	2	3
4	5	6	7	8	9	10	11	12	1	2	3	4
5	6	7	8	9	10	11	12	1	2	3	4	5
6	7	8	9	10	11	12	1	2	3	4	5	6
7	8	9	10	11	12	1	2	3	4	5	6	7
8	9	10	11	12	1	2	3	4	5	6	7	8
9	10	11	12	1	2	3	4	5	6	7	8	9
10	11	12	1	2	3	4	5	6	7	8	9	10
11	12	1	2	3	4	5	6	7	8	9	10	11
12	1	2	3	4	5	6	7	8	9	10	11	12

We may verify that every $a \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ we have $12 + a = a + 12 = a$.

12. The integers on the face of the clock also form a monoid under multiplication with the following multiplication table.

*	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	2	4	6	8	10	12	2	4	6	8	10	12
3	3	6	9	12	3	6	9	12	3	6	9	12
4	4	8	12	4	8	12	4	8	12	4	8	12
5	5	10	3	8	1	6	11	4	9	2	7	12
6	6	12	6	12	6	12	6	12	6	12	6	12
7	7	2	9	4	11	6	1	8	3	10	5	12
8	8	4	12	8	4	12	8	4	12	8	4	12
9	9	6	3	12	9	6	3	12	9	6	3	12
10	10	8	6	4	2	12	10	8	6	4	2	12
11	11	10	9	8	7	6	5	4	3	2	1	12
12	12	12	12	12	12	12	12	12	12	12	12	12

In the clock multiplication monoid, 1 has the property that $1 \times a = a \times 1 = a$ for every $a \in \{1, 2, \ldots, 12\}$.

3.1.3 Group

A notable property missing from a monoid is the ability solve equations. As an example, take the 12-clock above. Which of these equations can be solved in the integers?

1. Solve for a in $a \times 3 = 7$.
2. Solve for b in $4 \times b = 8$.
3. Solve for c in $7 \times c = 11$.

We observe that some such equations have unique solutions, such as $4 \times b = 8 \implies b = 2$. However, to systematically solve such an equation we need to be able to find an inverse with respect to the operation of the monoid.

$$a \times 3 = 7$$
$$a \times 3 \times 3^{-1} = 7 \times 3^{-1}$$

But there is no integer, $3^{-1} \in \mathbb{Z}$, such that $1 = 3 \times 3^{-1}$. However, some monoids do have the property that every element is invertible. A monoid for which every element has a unique inverse is called a group. Essential to the idea of inverse is that an inverse be unique.

Definition 2 (Group)**.** $(G, \circ)$ *is called a* group *if*

1. $(G, \circ)$ *is a monoid.*
2. *Inverse:* $\forall a \in G \; \exists a^{-1} \in G$ *such that* $a \circ a^{-1} = a^{-1} \circ a = e$

Definition 3 (Abelian Group)**.** *If* G *is a group such that* $a \circ b = b \circ a$ *for all* $a, b \in G$*, then we call* G *an* Abelian *group.*

A group has an identity which is also the identity of the monoid, which we already know to be unique. See Theorem 1. But we can ask a similar uniqueness question about the inverse. Can an element $a \in S$ have two different inverses? No, but why?

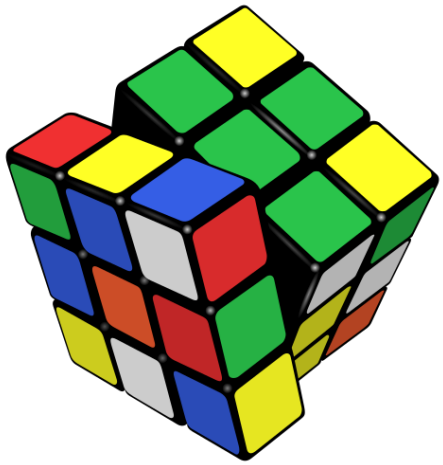


Figure 3.6: The set of rotations of the Rubik's cube is a group.

Theorem 2. *Each element of a group has exactly one inverse.*

Proof. Let G be a group with identity e. Let $a \in G$ have inverses x and y.

$$\begin{aligned} x &= x \circ e \\ &= x \circ (a \circ y) \\ &= (x \circ a) \circ y \\ &= e \circ y \\ &= y \end{aligned}$$

□

3.1.4 Examples of groups

1. The set of integers, $\mathbb{Z} = \{0, \pm 1, \pm 2, \ldots\}$, is a group under integer addition. Why?

   - $(\mathbb{Z}, +)$ is a monoid with 0 being the identity.
   - If $a \in \mathbb{Z}$ there exists $b \in \mathbb{Z}$ such that $a + b = 0$. E.g. $12 + (-12) = 0$

2. The integers under multiplication is not a group. Why?

3. The set of rotations of the $n \times n$ Rubik's cube is a group. Every rotation has an inverse rotation, and the null-rotation is the identity.

4. The set of 3×3 matrices of real numbers is not a group. Why?

5. The set of subsets of a given set using the operation of union is not a group. Why?

6. The set of subsets of a given set using the operation of intersection is not a group. Why?

7. The set subsets of a given set, G, using

$$A \circ B = (A \cap \overline{B}) \cup (\overline{A} \cap B)$$

as the operation, is a group. Every element is its own inverse. The identity element is the empty set, $\emptyset$.

8. Imagine a clock going from 1 to 11, rather than 1 to 12. Similar to the clock monoid seen above, here is the multiplication table of the strange 11-clock.

*	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	2	4	6	8	10	1	3	5	7	9	11
3	3	6	9	1	4	7	10	2	5	8	11
4	4	8	1	5	9	2	6	10	3	7	11
5	5	10	4	9	3	8	2	7	1	6	11
6	6	1	7	2	8	3	9	4	10	5	11
7	7	3	10	6	2	9	5	1	8	4	11
8	8	5	2	10	7	4	1	9	6	3	11
9	9	7	5	3	1	10	8	6	4	2	11
10	10	9	8	7	6	5	4	3	2	1	11
11	11	11	11	11	11	11	11	11	11	11	11

In this example, we see that the set is NOT a multiplicative group, because the element 11 has no inverse.

9. Every non-11 element in the table above has an inverse. We can tediously verify the existence of a unique inverse for every non-11 element.

   The set of non-11 elements of the 11-clock is a multiplicative group with the following multiplication table.

*	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	1	3	5	7	9
3	3	6	9	1	4	7	10	2	5	8
4	4	8	1	5	9	2	6	10	3	7
5	5	10	4	9	3	8	2	7	1	6
6	6	1	7	2	8	3	9	4	10	5
7	7	3	10	6	2	9	5	1	8	4
8	8	5	2	10	7	4	1	9	6	3
9	9	7	5	3	1	10	8	6	4	2
10	10	9	8	7	6	5	4	3	2	1

10. In Section 3.1.1, we defined the set $\mathcal{L}(\Sigma)$ of finite-length sequences using list concatenation as the monoid operation. This set is not a group. Why?

3.1.5 Ring

A monoid and a group are both sets with a single operation for which the elements of the set interact to obey a set of axioms (the monoid axioms or the group axioms). Now we will look at sets with two operations, which you can naïvely think of as addition and multiplication.

Definition 4 (Ring)**.** $(S, +, \times)$ *is called a* ring *if*

1. $(S, +)$ *is an Abelian group.*
2. $(S, \times)$ *is a monoid.*
3. *Distributive:* $a, b, c \in S \implies a \times (b + c) = (a \times b) + (a \times c)$.

In the special case that $(S, \times)$ *is commutative,* $(S, +, \times)$ *is called an Abelian ring.*

In the set of whole numbers we can think of multiplication as repeated addition. However, in many cases this does not hold. For example, matrix multiplication cannot be expressed simply as repeated matrix addition. In fact, in a ring, the only thing that connects the addition and the multiplication is the distributive axiom. Multiplication or addition may be defined in strange ways, but if the ring axioms are satisfied, the ring is valid.

In the integers $0 \cdot a = a \cdot 0 = 0$. Actually this principle, called *annihilation* is true for rings in general as shown in Theorem 3. Several other familiar connections between multiplication and addition also hold in general for rings. In particular $-1 \cdot -1 = 1$ (Theorem 4), and $-1 \cdot a = -a$ (Theorem 5).

Theorem 3. *For ring,* S *with additive identity* z *and multiplicative identity* e*, we have* $za = az = z$ *for all* $a \in S$.

Let's denote the additive inverse of any element x by $\overline{x}$

Proof.

$$\begin{aligned} za &= za + z \\ &= za + (za + \overline{za}) \\ &= (za + za) + \overline{za} \\ &= (z + z)a + \overline{za} \\ &= za + \overline{za} \\ &= z \end{aligned}$$

Likewise,

$$\begin{aligned} aa &= az + z \\ &= az + (az + \overline{az}) \\ &= (az + az) + \overline{az} \\ &= a(z + z) + \overline{az} \\ &= az + \overline{az} \\ &= z \end{aligned}$$

□

Theorem 4. *For ring, S, with multiplicative identity e, we have $\overline{e} \cdot \overline{e} = e$.*

Proof. Let z be the additive identity of the ring.

$$\begin{aligned} \overline{e} \cdot \overline{e} &= \overline{e} \cdot \overline{e} + z \\ &= \overline{e} \cdot \overline{e} + (\overline{e} + e) \\ &= \overline{e} \cdot \overline{e} + (\overline{e} \cdot e + e) \\ &= (\overline{e} \cdot \overline{e} + \overline{e} \cdot e) + e \\ &= \overline{e} \cdot (\overline{e} + e) + e \\ &= \overline{e}z + e \\ &= z + e && \text{By Theorem 3} \\ &= e \end{aligned}$$

□

Theorem 5. *For ring, S, with e, we have $\overline{e} \cdot a = \overline{a}$ for all $a \in S$.*

Proof.

$$\begin{aligned}
\overline{e} \cdot a &= \overline{e} \cdot a + a + \overline{a} \\
&= \overline{e} \cdot a + e \cdot a + \overline{a} \\
&= (\overline{e} + e) \cdot a + \overline{a} \\
&= z \cdot a + \overline{a} \\
&= z + \overline{a} && \text{By Theorem 3} \\
&= \overline{a}
\end{aligned}$$

□

3.1.6 Examples of rings

1. The integers $(\mathbb{Z}, +, \times)$ is a ring.
2. The integers $(\mathbb{N}, +, \times)$ is not a ring. Why?
3. The set of $n \times n$ matrices, for a fixed value of $n > 0$ is a ring.
4. The set of 2×3 matrices is not a ring. Why?
5. $\mathbb{Z}[x]$, the set of polynomials with integer coefficients such as $3x^4 + 2x^2 - 5x + 1$ is a ring.
6. $\mathbb{Q}[x]$, $\mathbb{R}[x]$, $\mathbb{C}[x]$, $(\mathbb{Z}/p)[x]$ for prime p.
7. The set of polynomials with rational coefficients with leading coefficient equal to 1 is not a ring. $(x^3 + \frac{1}{2}x^2 + 5x - \frac{2}{3})$ Why?
8. The set of subsets of a given set using intersection as multiplication and exclusive or as addition forms a ring.
9. The set of 8-bit sequences (bytes) consisting of 0 and 1, such as 00101101, with the two operations AND and XOR, defined bitwise using the following tables is a ring.

XOR	0	1
0	0	1
1	1	0

AND	0	1
0	0	0
1	0	1

For example

- 00101101 XOR 00110011 = 00011110
- 00101101 AND 00110011 = 00100001

3.1.7 Field

The most complicated structure we will examine is the field. You can think of a field as set which is a ring but more than that. In a ring you can add, subtract, and multiply. But in a field you can also divide with the exception that you cannot divide by zero.

Definition 5 (Field)**.** $(F, +, \times)$ *is called a* field *if*

1. $(F, +, \times)$ *is an Abelian Ring.*
2. $(F \setminus 0, \times)$ *is a (Abelian) group, where* 0 *is the identity under* $+$*.*

3.1.8 Examples of fields

1. The rational numbers, $\mathbb{Q}$.
2. The set of $n \times n$ matrices is not a field. Why?
3. The set of 2×2 matrices of the form $\begin{bmatrix} a & -b \\ b & a \end{bmatrix}$ where $a, b \in \mathbb{Q}$.
4. The integers modulo 12.
5. The integers modulo any prime such as 7.

$+$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

$\times$	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

With these addition and multiplication tables we can tediously verify that

- 0 is the additive identity.

- 1 is the multiplicative identity.
- Every element has an additive inverse.
- Every non-zero element has a multiplicative inverse.
- Addition and multiplication are associative.
- Addition and multiplication are commutative.
- The distributive property holds.

6. The set $\mathbb{Q}\{\sqrt{2}\} = \{a + b\sqrt{2} \mid a, b \in \mathbb{Q}\}$ is a field. Moreover, each of the following can be expressed in the same form $a + b\sqrt{2}$.

   - $(a_1 + b_1\sqrt{2}) + (a_2 + b_2\sqrt{2})$
   - $-(a + b\sqrt{2})$
   - $(a_1 + b_1\sqrt{2}) \times (a_2 + b_2\sqrt{2})$
   - $\frac{1}{a+b\sqrt{2}}$, provided a, b are not both 0.

   In particular

$$(a_1 + b_1\sqrt{2}) + (a_2 + b_2\sqrt{2}) = \underbrace{(a_1 + a_2)}_{\in\mathbb{Q}} + \underbrace{(b_1 + b_2)}_{\in\mathbb{Q}}\sqrt{2}$$

$$-(a + b\sqrt{2}) = \underbrace{-a}_{\in\mathbb{Q}} + \underbrace{-b}_{\in\mathbb{Q}}\sqrt{2}$$

$$(a_1 + b_1\sqrt{2}) \times (a_2 + b_2\sqrt{2}) = \underbrace{(a_1a_2 + 2b_1b_2)}_{\in\mathbb{Q}} + \underbrace{(a_1b_2 + a_2b_1)}_{\in\mathbb{Q}}\sqrt{2}$$

$$\frac{1}{a + b\sqrt{2}} = \underbrace{\frac{a}{a^2 - 2b^2}}_{\in\mathbb{Q}} + \underbrace{\frac{-b}{a^2 - 2b^2}}_{\in\mathbb{Q}}\sqrt{2}$$

   Why do we know $a^2 - 2b^2 \neq 0$ and $a + b\sqrt{2} \neq 0$?

3.1.9 Summary of Algebraic Structures

1. A *monoid* is a set where we can add.
2. A *group* is a set where we can add and subtract.
3. A *ring* is a set where we can add, subtract, and multiply.
4. A *field* is a set where we can add, subtract, multiply, and divide.

For simplicity we omit the formal definition of *semi-ring* which is a set in which we can add and multiply, but not necessarily subtract and divide.

These summaries are vague. For example when we say "add" in a monoid, the actual multiplication may be addition, multiplication, concatenation, or many other operations depending on the actual monoid. The distinction between adding and multiplying requires that both exist, thus they can only be distinguished in a semi-ring.

If a set has both a 1 and a 0 then dividing by 0 does not make sense. For example, in a field, we require all non-zero elements to have and multiplicative inverse even though all elements have an additive inverse.

3.1.10 Matrix Arithmetic

Two matrices of the same dimensions can be added. To compute the sum of two matrices, A, B, simply add the corresponding elements as shown in Equation (3.1)

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mn} \end{bmatrix}$$

$$c_{ij} = a_{ij} + b_{ij} \tag{3.1}$$

Two matrices, A, B, can be multiplied under the condition that the number of number of columns of A is the same as the number of rows of B. I.e., if A has dimension $m \times n$ (m rows and n columns), and B has dimension $n \times p$ (n rows and p columns), then $A \times B$ has dimension $m \times p$ (m rows and p columns). The formula for the i, j element of the product is given in equation (3.2)

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix}$$

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in} + b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj} \tag{3.2}$$

Each entry in the product, $C = A \times B$, can be thought of as a row of A times a column of B.

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \times \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_3 \end{bmatrix} = \begin{bmatrix} a_1 \cdot b_1 + a_2 \cdot b_2 + \cdots + a_n \cdot b_n \end{bmatrix}$$

3.1.11 Fast Exponentiation in a Monoid and Group

Recall the definition of exponentiation in Equation (2.6). In any monoid with operation $\circ$ and identity element e, we can compute an integer power of an element more efficiently according to the following formula.

$$x^n = \begin{cases} e & ;\text{if } n = 0 \\ x & ;\text{if } n = 1 \\ x \circ x^{n-1} & ;\text{if } n \text{ is odd} \\ (x^{\frac{n}{2}}) \circ (x^{\frac{n}{2}}) & ;\text{if } n \text{ is even} \end{cases} \tag{3.3}$$

If the monoid is also group, then inverse elements exist, so we can talk about *negative* exponents in the sense that $x^{-n} = (x^{-1})^n = (x^n)^{-1}$, in which case we can compute positive and negative powers efficiently. Either we compute $x^{|n|}$ and then compute its inverse, or we can compute x^{-1} and then raise that to the power of $|n|$.

$$x^n = \begin{cases} e & ;\text{if } n = 0 \\ x & ;\text{if } n = 1 \\ x \circ x^{n-1} & ;\text{if } n > 0 \text{ is odd} \\ (x^{\frac{n}{2}}) \circ (x^{\frac{n}{2}}) & ;\text{if } n > 0 \text{ is even} \\ (x^{-1})^{|n|} & ;\text{if } n < 0 \end{cases} \tag{3.4}$$

3.2 Programming perspective

3.2.1 Some new types

- `Option[T]`, `Some(x)`, `None`
- `Map[S,T]`
- Tuples `(S,T)`
- `X => Y`

3.2.2 Monoid

- `String` concatenation
- `List[Int]`, `Vector[Int]`, `Seq[Int]` concatenation
- Implement `fastPower` for `Double`.

- Refactor (or re-implement) `power` (for `String` and `List`) from Section 2.4.2 to work like `fastPower`.

3.2.3 Higher order functions

You will need functions like this to implement the Team Project in Section 3.3.2.

What is a *predicate*? It is a pure function which returns a `Boolean`. It is used to determine which a property holds for a given input value.

- `find` — find an element of a collection (if such exists) which satisfies a given predicate. `Option[T]`
- `forall` — does every element of a collection satisfy a given predicate. Remember the collection might be empty.
- `exists` — does at least one element of a sequence satisfy a given predicate.
- `filter` — collect all the elements of a collection which satisfy a given predicate.
- `sortBy`/`sortWith` — reorder (sort) the elements of a sequence according to a given function.
- `map` — create a new sequence by applying a given function iteratively to each element of a collection.

3.2.4 Multiple dimensional types

- `Seq[Seq[Int]]`, `Seq[Seq[Double]]`
- Implement matrix using `Vector[Vector[Double]]`. We implement `matrixIdentity`, `matrixAdd`, `matrixScale`, `matrixMultiply` using concentric loops.
- Then refactor to move the concentric loops into single function, `matrixTabulate`.
- Ring of matrixes (matrix add, subtract, multiply)

3.2.5 The `Map[]` type

- `Map[Int,Int]`, creating, manipulating
- Implement polynomial using `Map[Int,Double]`
- Ring of polynomials (polynomial add, scale, subtract, multiply)
- `almostEqual` comparing floating point

3.2.6 Binary Search

A binary search algorithm is a divide an conquer algorithm. Suppose we know that some *event* occurs at a number between x_{left} and x_{right}. For example, suppose $f(x) = 0$ for some $x_{left} < x < x_{right}$. Then if we take $x_{mid} = \frac{x_{left}+x_{right}}{2}$, then at least one of the following is true.

$$f(x) = 0 \qquad \text{for some } x_{left} < x \leq x_{mid} \tag{3.5}$$

$$f(x) = 0 \qquad \text{for some } x_{mid} < x < x_{right} \tag{3.6}$$

This means if we split the interval (x_{left}, x_{right}) in half, the *event* occurs either in the left half or the right half. In such a case, we can write a (recursive) function which divides the interval in half, and *recurs* either on the right half or left half. We end the recursion when we have found the event.

If we don't find the exact value we are searching for, then we must have a termination condition to avoid looping forever. Typically the way this is done is to recur until the interval is sufficiently small: i.e., until $x_{right} - x_{left} < \varepsilon$.

3.2.7 Testing

- Testing with `main`
- Debugging with print
- Property based testing
- Generating randomly selected object.

3.2.8 Team work

- projects using git and GitHub
- pull request
- merge and rebase

3.3 Coding Exercises

3.3.1 Compute number of combinations

Recall the definition

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}\,.$$

Write function to compute this quanity two different ways.

1. Compute it directly by computing the three factorials and performing the integer division.

2. Rather than computing $n!$ directly, express it as a list of prime factors. *I.e.*,

   - implement a function which, given n, uses recursion to compute and return the list of prime factors of n.
   - Use that function to implement a function which, given n, returns the prime factors of $n!$.
   - Effectively perform the division by removing all the prime factors in the denominator, $k!(n-k)!$, from those of the denominator, $n!$.
   - Finally, multiply the remaining factors to compute $\binom{n}{k}$.

E.g.,

$$\begin{aligned}
\binom{7}{3} &= \frac{7}{3!(7-3)!} \\
&= \frac{7}{3! \cdot 4!} \\
&= \frac{7 \cdot 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2}{(3 \cdot 2)(4 \cdot 3 \cdot 2)} \\
&= \frac{7 \cdot \cancel{3} \cdot \cancel{2} \cdot 5 \cdot \cancel{2 \cdot 2 \cdot 3 \cdot 2}}{(\cancel{3} \cdot \cancel{2})(\cancel{2 \cdot 2 \cdot 3 \cdot 2})} \\
&= 7 \cdot 5 \\
&= 35
\end{aligned}$$

Compare the two functions both in terms of computation time, and also find values of $\binom{n}{k}$ which are computable by one function but not the other because of integer overflow.

3.3.2 Team Project: Structure Recognition

- Implement functions to detect monoid, group, ring, and field given a finite set, and the $+$ and $\cdot$ operations.
- When finished with your function, commit, push, and send a pull request.
- To test your code, you will need to pull the changes in the repository.
- Define $\mathbb{Z}/n$, the integers modulo n. Is it a monoid? Is it a group (Abelian)? etc. ...
- For which values of n is $\mathbb{Z}/n$ a field?

3.3.3 Square matrices of Double

- Implement matrix arithematic functions: add, scale, subtract, multiply, identity, etc..

- Implement power multiply for matrices: `matrixPower`. Do this in two different ways: by repeated multiplication (compute M^n as a result of $n-1$ multiplications), and again using the fast power algorithm.

- Can you devise a way to test whether `fastPower` is really faster?

- Use the `matrixPower` function to compute any three consecutive Fibonacci numbers: F_{n-1}, F_n, and F_{n+1},

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix}$$

  Do these results agree with those you computed in unit 1?

- In Section 2.4.3, you computing Fibonacci number using an exponentially complex algorithm. Now you have a way which you suspect is more efficient (faster). Devise an experiment to determine which of the approaches is faster in terms of execution time? Hint, you may need to contstruct a loop to do the same computation 100 times or 1000 times, and the divide the total time by 100 or 1000 to compute the average time per computation.

3.3.4 Group Interactive Project

- Implement the polynomial ring, with addition, subtraction, multiplication.

- What does it mean for two polynomials to be equal? For example: are these polynomials equal or not?

$$x^3 - 2x + 1 \tag{3.7}$$
$$2x^3 - 4x + 2 \tag{3.8}$$
$$-x^3 + 2x - 1 \tag{3.9}$$

  Should we consider a polynomial to be the function itself, or should we consider it as an equivalence class of all polynomials with the same roots. I.e., can we consider *normalized* polynomials such that the leading term is 1?

- Implement `polyAlmostEqual`. Why do we use `polyAlmostEqual`? Can we detect whether two polynomials are exactly equal?

- Every odd degree polynomial has a real root. Can you justify this claim? Write a function which uses a binary search to find (estimate) at least one real root of a given odd-degree polynomial.

- Can you implement `polynomialPower` using `fastPower`. Is it faster than an iterative implementation?

- Given a list of roots (perhaps with repeated roots), generate the polynomial. Do this in two ways:

  1. Use the polynomial multiply function written above to expand

  $$(x - r_1)(x - r_2) \dots (x - r_{n-1})(x - r_n) .$$

  2. Generate the expansion directly by sums of products of subsets. If

  $$\Pi_{i=1}^{n}(x - r_i) = \Sigma_{i=0}^{n} a_i x^{n-i}$$

  and if

  $$S = \{T \in \{r_1 \dots r_n\} \mid \ |T| = n\} ,$$

  then

  $$a_k = \begin{cases} 1 & ; \text{if } k = 0 \\ \Sigma_{T \in S} \Pi_{k \in T} k & ; \text{if } 1 \leq k \leq n \end{cases} \tag{3.10}$$

- Implement a function to generate a randomly selected polynomial of a given degree: `randomPoly`.

Discussion: Is the set of polynomials of degree n with integer coefficients an infinite set or finite set? Is this set closed under addition? Are multiplication and addition associative and commutative? Is it closed under multiplication? Does this set form a monoid, group, ring, or field?

How does the answer change if addition and multiplication of coefficients is done in $\mathbb{Z}/(n+1)$.

How would you generalize to the polynomial ring over a given field?

Consider the set of polynomials of degree n whose scalars are the field $\mathbb{Z}/p$. How would you implement addition, subtraction, and multiplication?

What if arithmetic on not only coefficients but also exponents is done in $\mathbb{Z}/(n+1)$.

What would you need to do to use the library you've written `isMonoid`, `isGroup` etc to test your hypotheses? How would you write a program to generate all such polynomials?

Discussion: Is the set of $n \times n$ matrices with integer content a finite or infinite set? What if we restrict the content to $\mathbb{Z}/(n+1)$? How would you need to change the matrix manipulation code you've already written to accommodate arithmetic in $\mathbb{Z}/(n+1)$?

3.3.5 Complex matrices [optional]

The complex numbers, $\mathbb{C}$, are an extension of the real numbers which allows us, among other things, to represent $i = \sqrt{-1}$ and consequently to represent roots of polynomials such as $x^2 + 1 = (x + i) \times (x - i)$. A complex number, can be expressed as $a + bi$ with a being the real part and b being the imaginary part. Together $a + bi$ is a complex number.

An alternate way of representing a complex number is as a 2×2 matrix.

$$a + ib = \begin{bmatrix} a & -b \\ b & a \end{bmatrix}$$

- Verify anecdotally by constructing 4 concentric loops that

  $$(a_1 + ib_1) \times (a_2 + ib_2) = (a_2 + ib_2) \times (a_1 + ib_1)\,.$$

  Use the matrix multiply function you implemented previously.

- Verify anecdotally that $i^2 = -1$. What does this mean in terms of the matrix representation?

Discussion: in which cases regarding matrices and complex numbers can we check associativity and commutativity? Is matrix addition

commutative? Is matrix multiplication commutative? Is complex multiplication commutative?

Discussion: If matrix multiplication is not commutative, but complex multiplication is, why can we implement complex numbers in terms of matrices?

3.3.6 Monoid Exponentiation

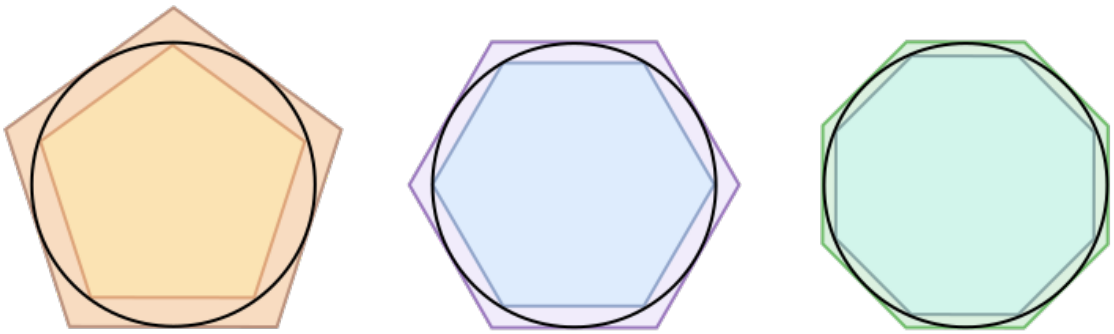
You have implemented a *fast-power* function several times for different types. Here we implement the same thing for a monoid, and then refactor the previously implemented functions to use `monoidFastPower`.

- Implement `monoidFastPower`. The challenge is to figure out what the arguments of the function are.
- Refactor exponentiation of `Double`.
- Refactor matrix exponentiation.
- Refactor polynomial exponentiation.
- Refactor complex exponentiation.

Discussion: What must you do to allow negative exponents? Is it possible for a monoid? Which algebraic structure is necessary to support negative exponents

Chapter 4

Convergence



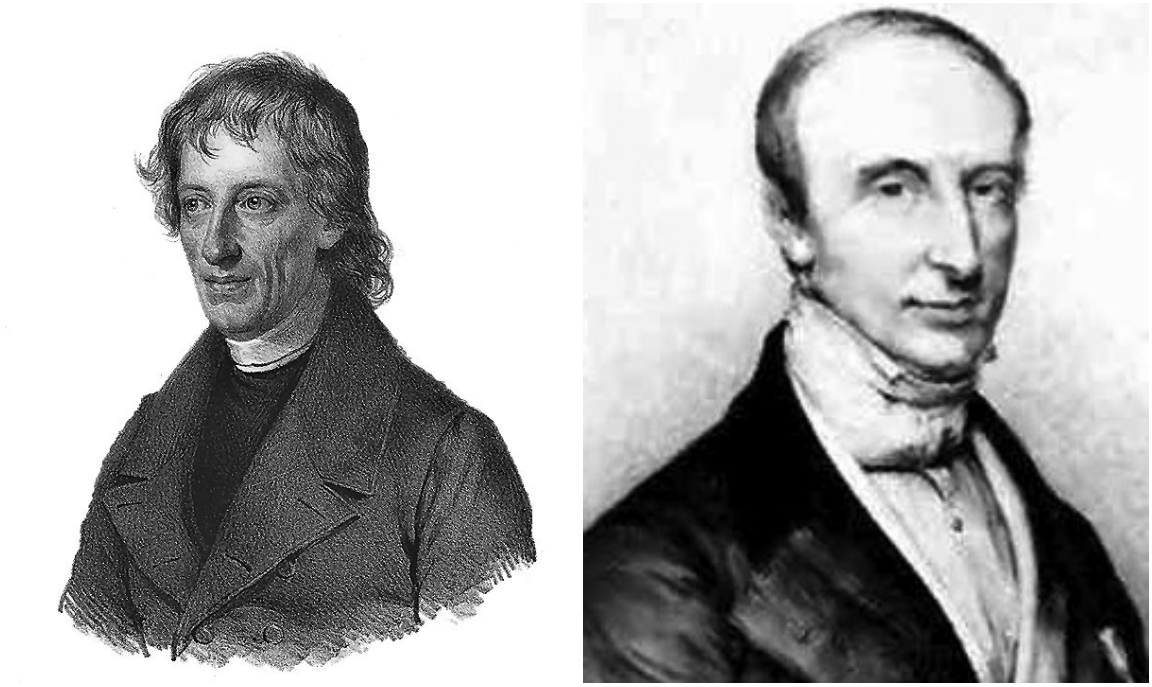
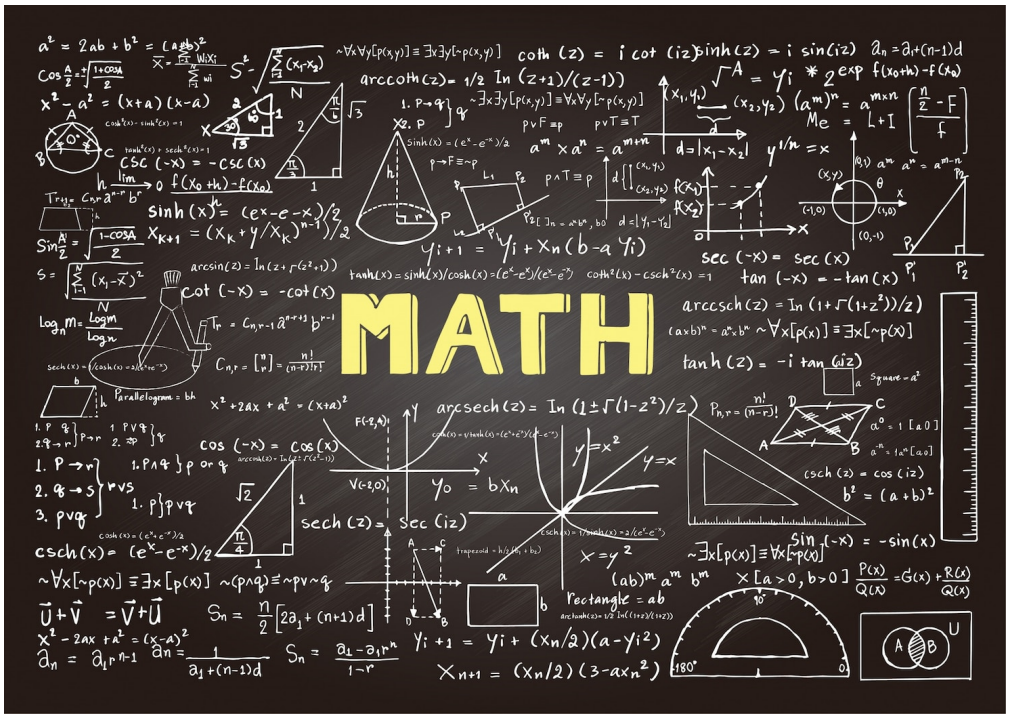


Figure 4.1: Bernard Bolzano (left), Augustin-Louis Cauchy (right)

In this section we will talk about a characteristic property/attribute which distinguishes the rational numbers, $\mathbb{Q}$, from the real numbers, $\mathbb{R}$. That important feature of $\mathbb{R}$ not shared by $\mathbb{Q}$ is *convergence*. But what do we mean by convergence?

We will discuss the mathematical definition of convergence, and we will write some programs which approximate real quantities with sequences which approximate them.

Isaac Newton and Gottfried Leibniz invented calculus around 1660 or 1670. Some 150 years later, around 1820 Augustin-Louis Cauchy and Bernard Bolzano presented the concept of the limit. Thanks to the work of Cauchy and Bolzano and his contemporaries, we can understand at a fundamental level, the work of Newton and Leibniz.



4.1 Mathematical perspective

In the previous sections we talked about the natural numbers, $\mathbb{N}$, the integers, $\mathbb{Z}$, the rationals, and $\mathbb{Q}$. In particular we talked about what distinguishes them from each other.

In an intuitive sense:

- $\mathbb{N}$ is a set where we can add and multiply;
- $\mathbb{Z}$ is a set where we can add, subtract, and multiply; and
- $\mathbb{Q}$ is a set where we can add, subtract, multiply, and divide.

Unfortunately, we don't have a good example of **numbers** where we can add but not multiply; however, we saw examples of so-called monoids such as simple sets where the union operation is associative, the empty set, $\emptyset$, is the identity element, but there is no inverse operation of union, and nothing like multiplication.

4.1.1 Intuition of convergence?

In the figure on page 66 we can convince ourselves that the circumference of the circle is a fixed quantity. I.e., it is not an vague, indefinite concept, but the circumference of a circle has a real, actual, definite length.

The length of the circumference of the circle is always greater than the length of the perimeter of an inscribed polygon, and is always less than the length of the perimeter of a circumscribed polygon. Moreover, we have a decreasing sequence of numbers and an increasing sequence of numbers which grow arbitrarily close together. I.e., choose any small number, ε. Now find a sufficiently large n such that the perimeter of the circumscribed n-sided polygon is within ε of the perimeter of the inscribed polygon. The sandwiched value is the circumference of the circle.

The real numbers, $\mathbb{R}$, is the smallest set which contains all these numbers.

$$\mathbb{Q} \subset \mathbb{R}$$

$$\mathbb{R} \not\subset \mathbb{Q}$$

As another example, using infinite sums which we will talk about in a later unit, consider the sequence: $(a_0, a_1, a_2, \ldots)$, such that a_n is defined as:

$$a_n = \sum_{k=0}^{n} \frac{8}{(4k+1)(4k+3)}$$

This sequence is discussed on wikipedia.[1]. This is a sequence of rational numbers; each numerator is an integer, namely 8, and each denominator is an integer. The successive rational numbers, a_n and a_{n+1} may be made as close as you like by choosing sufficiently large n. Thus this sequence is Cauchy. Nevertheless, the value this sequence tends to is not rational. If you chose any rational number, $q \in \mathbb{Q}$, then either:

1. You can find an N such that if $n > N$ then $a_n > q$.

2. You can find an N such that if $n > N$ then $a_n < q$.

[1] Leibniz formula, `https://en.wikipedia.org/wiki/Leibniz_formula_for_pi`

Thus we say that this sequence does not converge in $\mathbb{Q}$. However, it converges to in $\mathbb{R}$. In particular

$$a_n \to \pi \, .$$

4.1.2 The limit of a sequence

The limit is the most fundamental concept in Calculus.

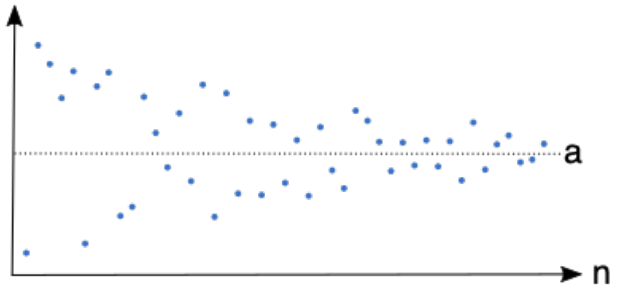
$$\lim_{n\to\infty} a_n = a$$

means that $\forall\ \varepsilon > 0\ \exists\ N \in \mathbb{N}$ such that if $n > N$ then $|a_n - a| < \varepsilon$.

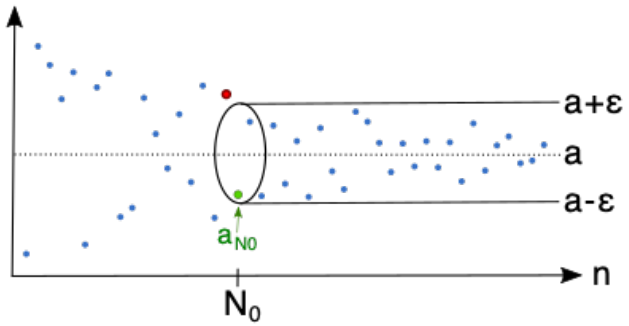
A sequence $\{a\}_n$ converges to a, written $a_n \to a$, if

$$\lim_{n\to\infty} a_n = a \, .$$

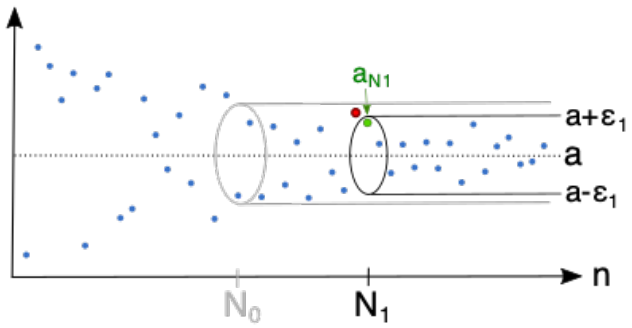
A sequence is $\{a\}_n$ is said to be Cauchy, if $\forall\ \varepsilon > 0\ \exists\ N \in \mathbb{N}$ such that if $n > N$ and $m > N$ then $|a_n - a_m| < \varepsilon$.



Example of a sequence which converges to the limit a.



Regardless which $\varepsilon > 0$ we have, there is an index N_0, so that the sequence lies afterwards completely in the epsilon tube $(a - \varepsilon, a + \varepsilon)$.



There is also for a smaller $\varepsilon_1 > 0$, an index N_1, so that the sequence is afterwards inside the epsilon tube $(a - \varepsilon_1, a + \varepsilon_1)$.

4.2 Programming perspective

1. tail recursion,
2. fixed point
3. higher order functions
4. function type
5. representing finite and infinite sequences.
6. sequence functions: `map`, `filter`, `max`, `sum`, `head`, `tail`,
7. `for` comprehension

4.3 Coding Exercises

4.3.1 Sequences

- Define a sequence

$$a_n = \sum_{k=0}^{n} \frac{8}{(4k+1)(4k+3)}\,.$$

  Given n, compute the sequence of $n+1$ numbers, and use the `sum` method to add them up. (We will discuss a better way tomorrow).

- Compute the same sums using normal recursion and tail recursion.

- Convince yourself that $\{a_n\}$ is strictly increasing; i.e., $a_n < a_{n+1}$. Can you show this empirically? Can you prove this mathematically?

- However, the sequence of successive differences is decreasing: if $d_n = a_{n+1} - a_n$ then the sequence $\{d_n\}$ is decreasing: $d_n > d_{n+1}$. Show this empirically and mathematically.

4.3.2 Convergence

- Now implement a function such that given ε, find and return N, such that

$$n > N \implies |a_n - \pi| < \varepsilon\,.$$

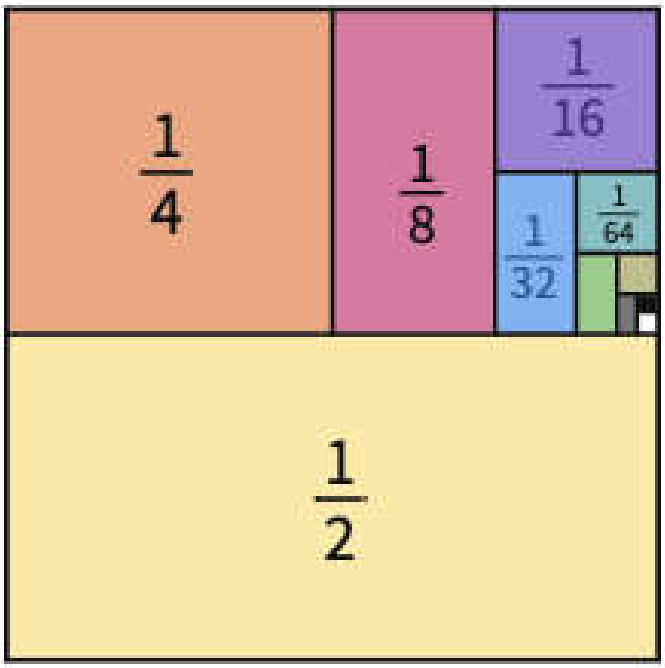
- Detect Cauchy of sequences: write a function such that given a sequence in the form of a function, a of an `Int` k, and a `Double` ε. The function should return an `Int`, k, indicating the first value for which $|a_k - a_{k+1}| < \varepsilon$.

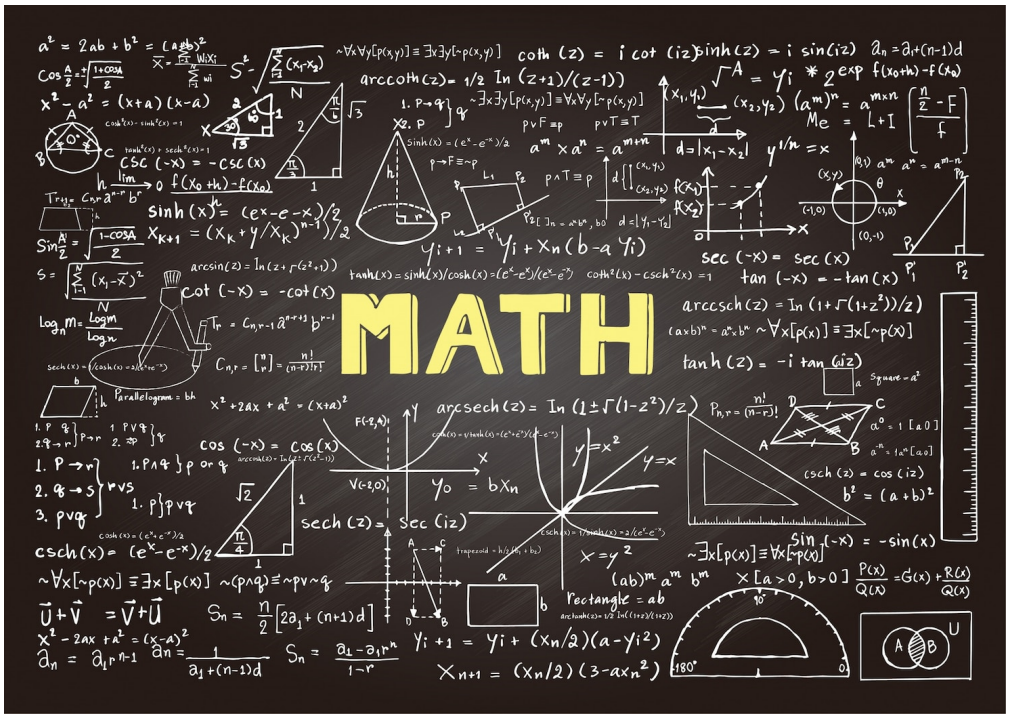
- Use Google or Wikipedia etc, to find another sequence which converges to π. Use the function you just wrote to empirically verify this conversion.

- Now you have two sequences which both converge to π. Does the sequence created by averaging the corresponding elements also converge? If so, what does it converge to? Does it converge faster or slower? Can you devise an experiment to show empirically?

Chapter 5

Infinite sums





5.1 Mathematical perspective

In this chapter we'll look at ways to compute sums like these

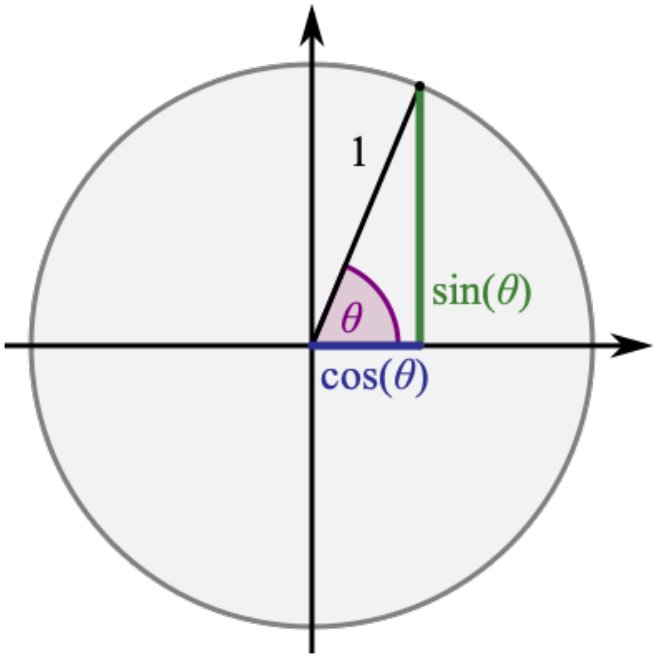
$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!} \tag{5.1}$$

$$\sin(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!} \tag{5.2}$$

$$\cos(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!} \tag{5.3}$$

$$\ln(1+x) = \sum_{k=1}^{\infty} \frac{(-1)^{k-1} x^k}{k}\,. \tag{5.4}$$

1. Review functions sin, cos, exp, and ln.

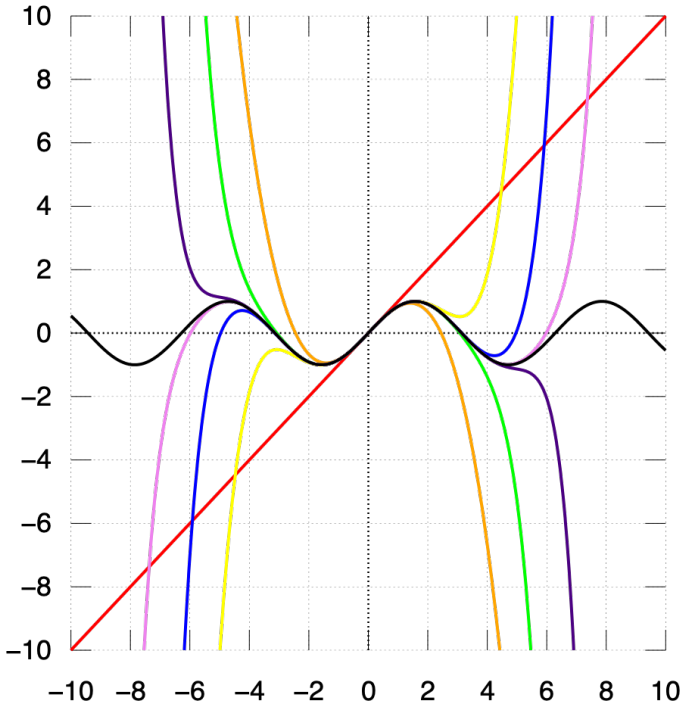


2. Contrast convergence and summability.

3. Taylor series is of the form

$$\sum_{k=0}^{\infty} a_n x^n \, .$$

In some cases this series converges to an interesting function. Since the x^n terms are increasing rapidly, the a_n terms must therefore decrease (in absolute value) rapidly otherwise the series will not converge (it will diverge).



As the degree of the Taylor polynomial rises, it approaches the correct function. This image shows $\sin x$ and its Taylor approximations by polynomials of degree 1, 3, 5, 7, 9, 11, and 13 at $x = 0$.

5.2 Programming perspective

1. `Option[]`
2. folding: `foldLeft`, `reduce`, `reduceOption`
3. `Either[]`
4. Introduce `foldM`

5.3 Coding Exercises

5.3.1 Taylor series with numbers

1. Reimplement factorial using `foldLeft`.

2. Implement the sin, cos, and exp functions in terms of the infinite Taylor series, understand conditions of convergence, avoiding infinite loops.

3. Devise a way using an ε and the `foldM` function to elegantly terminiate the iteration with the target value is close enough.

4. Improve convergence using shifting and scaling.

$$\sin(x) = \sin(x - 2\pi k) \tag{5.5}$$
$$\cos(x) = \cos(x - 2\pi k) \tag{5.6}$$

To improve the convergence of sin and cos, translate the argument some number of full periods so that the argument lies between $-\pi$ and π. Choose k such that $-\pi \leq x - 2\pi k \leq \pi$, then compute $\sin(x - 2\pi k)$ or $\cos(x - 2\pi k)$

$$e^x = (e^{x/k})^k \tag{5.7}$$

To improve the convergence of e^x, compute the exponential of a smaller power, then raise the result to an integer power. Choose k such that $x/k < 2$. Compute $e^{x/k}$. Then use the fast-power algorithm to raise that value to the power of k.

5.3.2 Taylor series with matrices

1. Compute sin and cos of square matrices.

2. Determine experimentally the value of $\cos^2(M) + \sin^2(M)$ for a square matrix M.

3. For square matrices, x and y, can you anecdotally verify or disprove the identity $e^x \times e^y = e^{x+y}$?

4. Can you find other trigonometric identities which have an analog in the world of trigonometry on matrices? Can you find identities on the real numbers which don't correspond to matrices?

5. Compute sin and cos of complex numbers using 2×2 matrix.

6. Can you anecdotally verify or disprove the identity $e^x \times e^y = e^{x+y}$?

7. For complex numbers, x and y, can you anecdotally verify or disprove the identity $e^x \times e^y = e^{x+y}$?

Chapter 6

Differential Calculus [Optional]

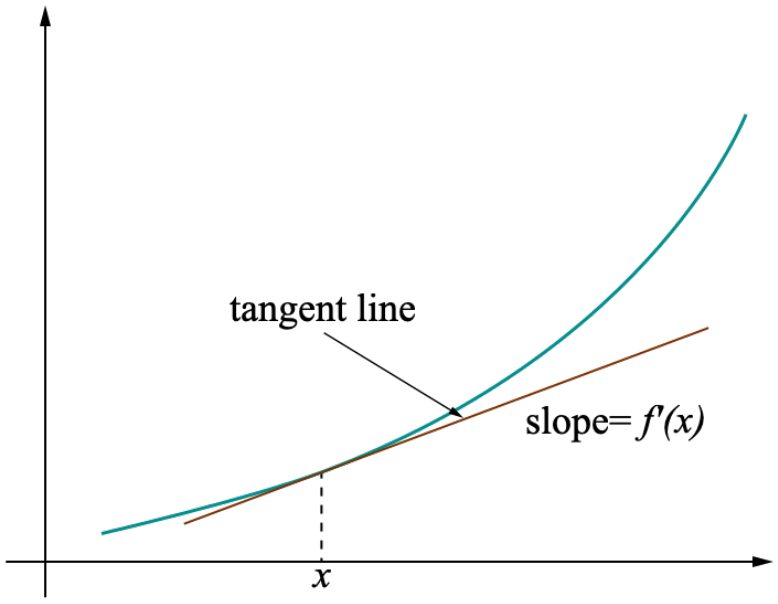
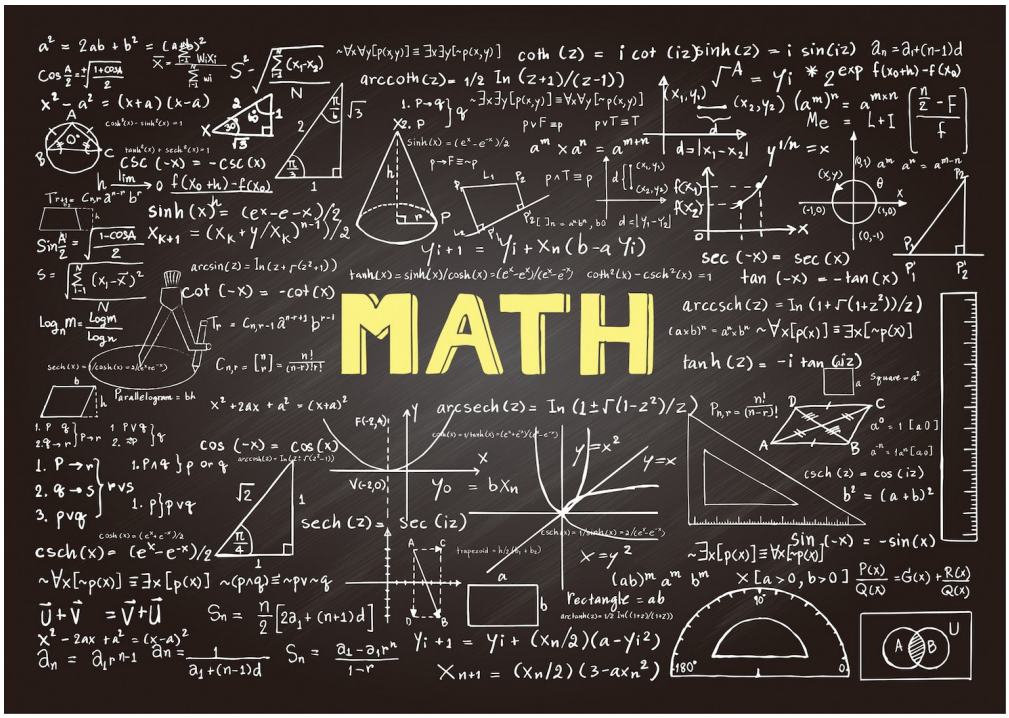


Figure 6.1: Intuitively, the derive compute the slope of the line tangent to a curve at a given point.



6.1 Mathematical perspective

We start our discussion of the derivative by talking about the limit. We saw the limit already in terms of sequences. Now we look at a slightly different concept, and extension, of the limit in terms of a function of a single variable. We will thereafter, define the derivative in terms of the limit. Additionally, we will follow the same model programmatically, by defining a function to estimate the limit, then to implement a function to estimate the derivative using the limit function.

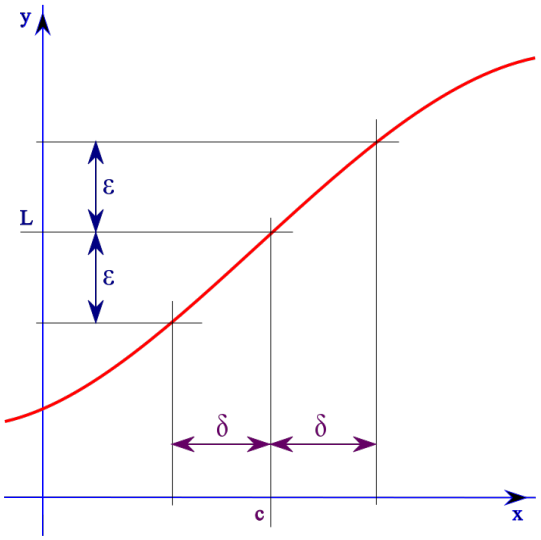
1. Intuition of limit. Consider the function:

$$f(x) = \frac{x-1}{x^2-1}\,.$$

We cannot evaluate this function at $x = 1$; however if we take values of x close to 1, the function evaluates just fine, and is well-behaved. Take a sequence of values approaching 1, and ask yourself what value does $f(x)$ approach?

It turns out that any sequence $a_n \to 1$ (except those for which $a_k = 1$ for some k) corresponds to a convergent sequence $f(a_n) \to \frac{1}{2}$.

Another way of phrasing this intuition is that we can make $\frac{x-1}{x^2-1}$ close to $\frac{1}{2}$ by making x close to 1. But that statement is a vague. We can make the statement more precise by saying that we can make $\frac{x-1}{x^2-1}$ arbitrarily close to $\frac{1}{2}$ by making x sufficiently close to 1.



2. Definition of limit

   What does this notation mean?

$$\lim_{x \to a} f(x) = L$$

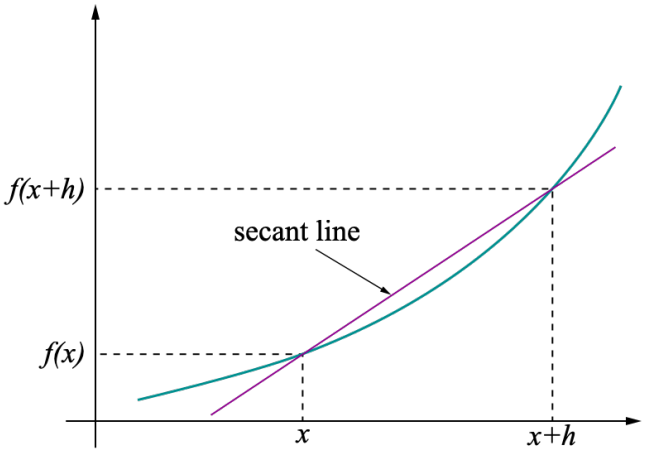
   It means that: $\forall \varepsilon > 0$ $\exists \delta > 0$ such that

$$|x - a| < \delta \implies |f(x) - f(a)| < \varepsilon \,.$$

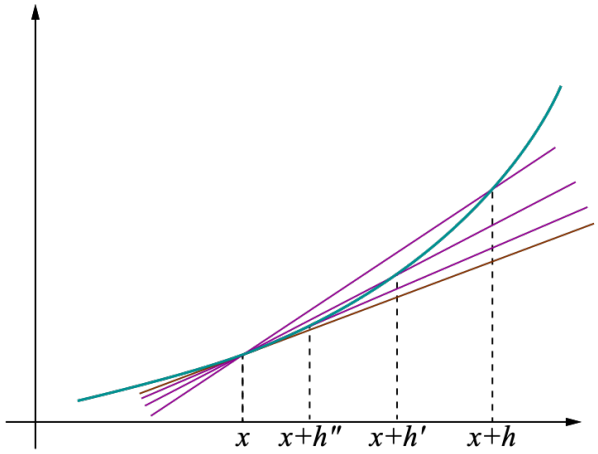
   Using the language we used above: the limit of a function f at a is L means that we can make the value of f arbitrarily close to L by making its argument sufficiently close to a.

3. Intuition of derivative

   Again we can generate a sequence of values (of $h \to 0$) approaching 0, and from that we can generate a sequence of values of the slope of f, i.e., $\frac{f(x+h)-f(x)}{h}$. Does this sequence of slopes converge when the sequence of h values converges to 0?



The secant to curve $y = f(x)$ determined by points $(x, f(x))$ and $(x + h, f(x + h))$.



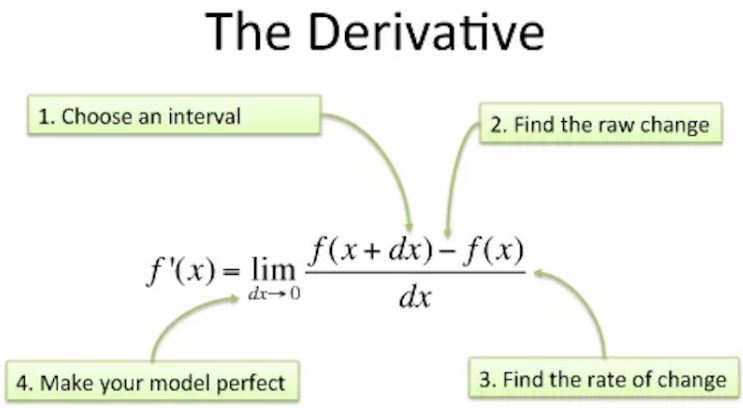
The tangent line as limit of secants.

Figure 6.2: We may estimate the derivative by a sequence of secant lines which estimate the tangent line. The secant lines approach the tangent line in Figure 6.1.

If $h \to 0$ then does $\frac{f(x+h)-f(x)}{h}$ approach some value? The value may very well be different for each x. If so that set of values is therefore a function of x, which leads to the definition of the derivative.

4. Definition of derivative

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h} = \lim_{a\to x} \frac{f(x)-f(a)}{x-a}\,.$$



5. Linearity of derivative.

$$f(x) = h(x) + g(x) \implies f'(x) = h'(x) + g'(x)\,,$$

and

$$f(x) = \alpha h(x) \implies f'(x) = \alpha h'(x)\,.$$

6. Product rule

$$f(x) = h(x)g(x) \implies f'(x) = f(x)g'(x) + f'(x)g(x)\,.$$

7. Derivative of polynomial using induction and product rule.

$$f(x) = x^n \implies f'(x) = nx^{n-1}\,.$$

8. Thus we can compute the derivative of any polynomial: If

$$f(x) = \sum_{k=0}^{n} a_n x^k\,,$$

then

$$f'(x) = \sum_{k=0}^{n} k a_n x^{k-1}\,.$$

6.2 Programming perspective

1. Definition of derivative in terms of limit,
2. Derivative of polynomial

6.3 Coding Exercises

1. Implement the `limit` function as a function which manipulates functions.

2. Use the `limit` function to evaluate

$$\lim_{x \to 1} \frac{x - 1}{x^2 - 1}.$$

3. Compute derivative by successive approximation using the previous implementation of limit.

4. Compute exact derivative of a polynomial

5. Compare two approaches numerically.

6. Compute cos in terms of derivative of sin,

7. Compare to Taylor series computation.

Chapter 7

Integral Calculus [Optional]

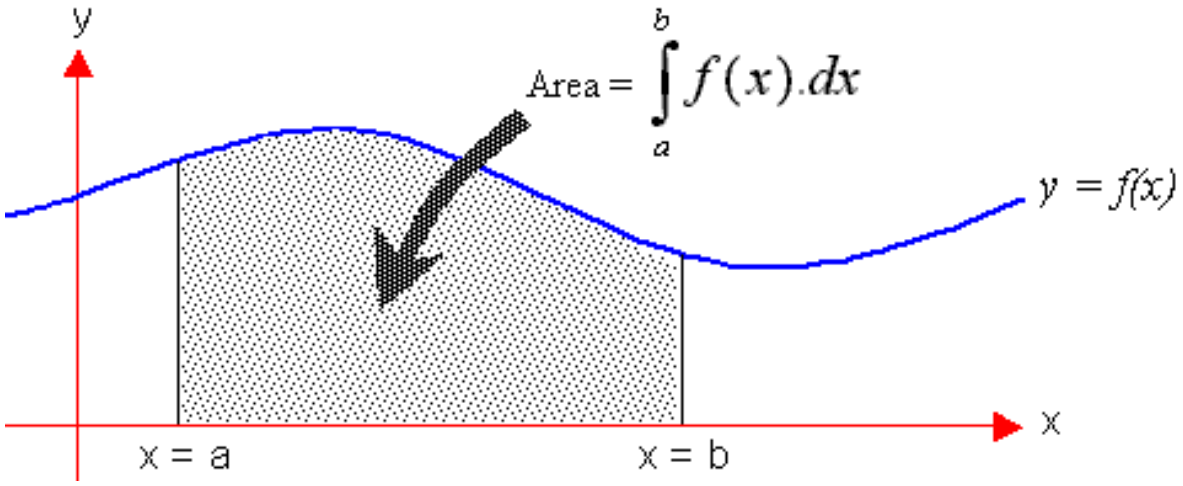
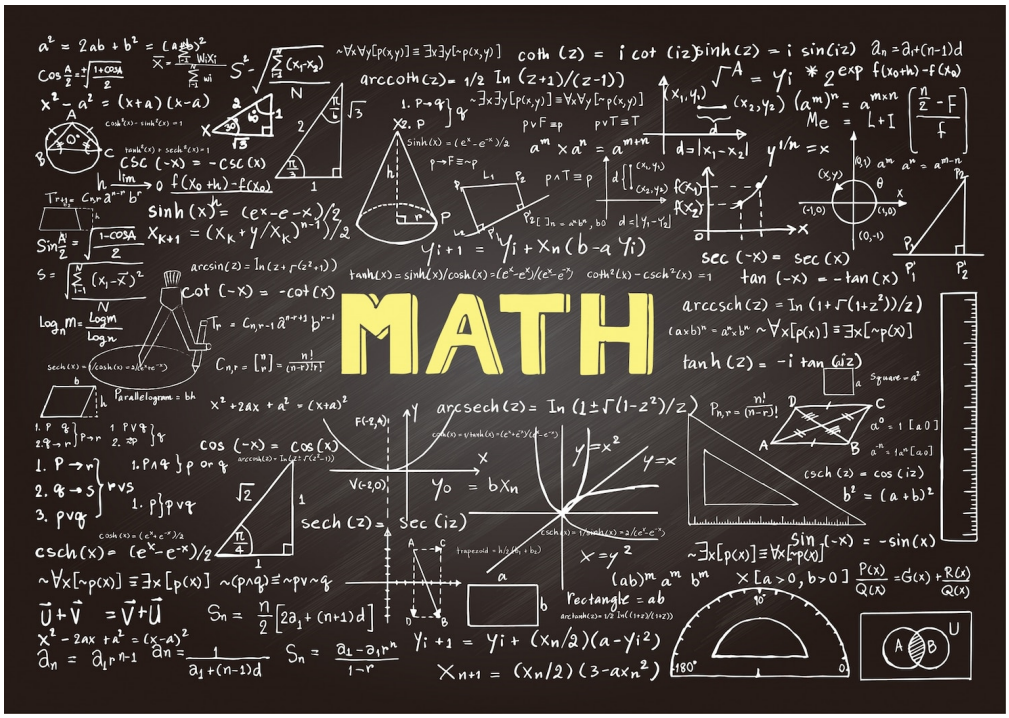


Figure 7.1: The initial intuition of the Riemann integral is a computation of the area *under* the curve of a given function.



7.1 Mathematical perspective

1. The notation and intuition of the integral.
2. Understanding the Riemann sums in terms of infinite sums of rectangles.
3. Integral of polynomial:

$$\int x^n \, dx = \frac{1}{n+1} x^{n+1}$$

So by linearity,

$$\int \sum_{k=0}^{n} a_k x^n \, dx = \sum_{k=0}^{n} \int a_k x^n \, dx \tag{7.1}$$

$$= \sum_{k=0}^{n} a_k \int x^n \, dx \tag{7.2}$$

$$= \sum_{k=0}^{n} \frac{a_k}{n+1} x^{n+1} . \tag{7.3}$$

4. Fundamental Theorem of Calculus:

   - The derivative of the integral is the function.

$$F(x) = \int_a^x f(t)dt \implies F'(x) = f(x) \tag{7.4}$$

$$\frac{d}{dx}\int_a^x f(t)dt = f(x)\,. \tag{7.5}$$

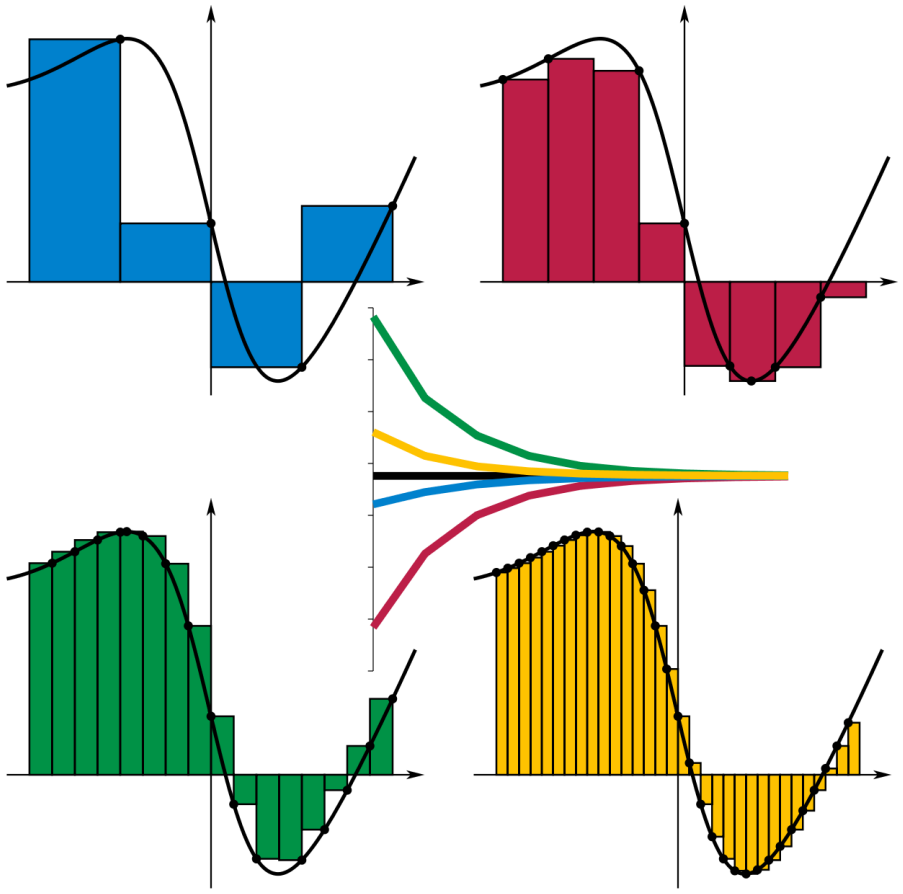
   - The integral of the derivative is the function.

$$F'(x) = f(x) \implies \int_a^b f(x)dx = F(b) - F(a) \tag{7.6}$$

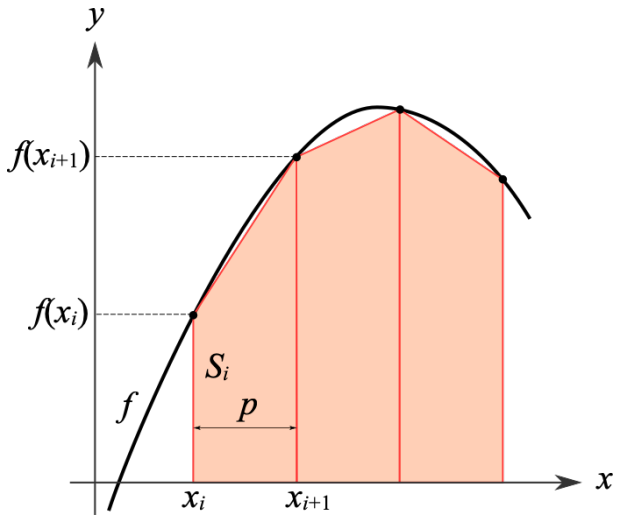
$$\int_a^x f'(t)dt = f(x) - f(a)\,. \tag{7.7}$$

7.2 Programming perspective

1. Expressing limit of sum of rectangles



2. Expressing limit of sum of trapezoids



7.3 Coding Exercises

1. Implement integral (function of function) using limit, sum of rectangles, and sum of trapezoids. Which converges faster?
2. Compute area under line, and show experimentally that it grows quadratically.
3. Compute area under parabola, and show experimentally that it grows cubically.
4. Compute area of a circle by integrating $\sqrt{1-x^2}$. I.e., compute
$$2\int_{-1}^{1}\sqrt{1-x^2}\,dx\,.$$
5. Compute explicit integral of polynomial
6. Verify the fundamental theorem of calculus experimentally.