Technological Fluency Labs for Math and Science Students, Version 0.1

Len Brin, Joe Fields, Younhee Lee and Ray Mugno

Southern Connecticut State University

Contents

Preface

The goal of this book is to help you learn about a variety of computational tools that will be useful in your career as a mathematician or scientist.

1 Introduction

This book aims to introduce the reader to several computational tools that are useful for people entering careers in Mathematics and the Sciences. We have chosen to emphasize systems that are open-source but we generally indicate other options in the introduction to each chapter.

The book is divided into 8 main chapters.

- This Introduction.
- Writing with embedded mathematical content.
- Computational Algebra.
- The creation and inclusion of Graphics in technical documents.
- Interactive Geometry.
- A chapter on the use of Spreadsheets.
- Statistics
- Miscellaneous

In each of those chapters there will be technological as well as mathematical learning goals. The point of this book is to collect laboratories/activities that will allow one to reach these goals through an active learning strategy: doing stuff, not being told about it!

An important aspect of developing a facility with tech is that you get better at it the more you do it. Many people advocate that "learning how to learn" is one of the biggest benefits of a college education. So we hope that part of the takeaway from doing these labs is that it will be easier for you to figure out other tools that you'll encounter

in the future. For example, a great many computer applications allow you to access their functionalities via a menu system. Many of those menu entries have a keyboard shortcut associated with them. Once you see the increased productivity you get from using such shortcuts, you start to look for them in other applications. And, there are fairly consistent conventions about how those shortcuts are selected so knowledge you gain in one app often transfers over directly to others!

In the early days of computing, someone introduced the initialism RTFM which doesn't really stand for "Read that fine manual." That's one piece of advice that you should certainly take away from the TFLabs experience – learn how to access and read the help facility! A couple of other suggestions for getting up to speed quickly in a new computing environment:

- Often, the help for a particular command will include examples. Scrolling past that wall of confusing text until you get to the relevant examples isn't a terrible strategy.
- See if there are so-called "tool-tips" – hovering your mouse over a button often causes a pop-up that displays a hint about what the button does.
- Look for the aforementioned keyboard shortcuts. A lot of programs use Ctrl-Z as a shortcut for "undo" (this can be super useful! Unless the command is Ctrl-U in your particular application.)
- Learn to read error messages. These messages are generated programattically when something goes wrong - this usually makes them a bit cryptic. If you're able to crack the code, you'll probably at least get a hint as to what caused the problem.
- Use your favorite search engine to see if there is relevant help already out there. "Hey Siri, how do I get an ampersand in LaTeX?" (I really don't know if that will work, but putting that question in my browser's search bar sure does!)
- Doing the previous "googling" will often lead you to online help fora like Quora, Reddit and Stack Overflow. Reading threads in a forum can be really helpful. Posting questions in a forum is hit-or-miss – sometimes you'll get a quick useful answer, other times the result is not what one would want...

Let's get started!

2 Writing

2.1 HTML

Have you ever seen a web URL like

`https://www.agnesscott.edu/lriddle/women/love.htm`

and wondered what the `https` or the `htm` meant? In both `https` and `htm`, the `ht` is short for hypertext. The `tp` in `https` is short for transfer protocol and the `s` is for secure. The `m` in `htm` is for markup. All webpages, when it comes down to it, are made of HTML (hypertext markup language) code. The HTML tells the browser what should be displayed and, in general terms, the desired layout. The browser takes care of the fine details depending on the type of device, size of the screen, or personal preferences set by the user.

Markup in HTML is like a note to the browser telling the browser how to display the following content. The markup won't be displayed, but it will modify how the content will be displayed. Take this screenshot from `https://www.southernct.edu/` for instance.

BACK TO CAMPUS FALL '21 #SouthernSTRONG

As the university prepares for the start of the fall 2021 semester, we remain #SouthernStrong, with all of the following academic offerings and campus services in place for our students in a safe, engaging campus environment. Our goal is to get you off to a great start, whether you're returning to campus or joining our community for the first time!

Simplified just a bit, the HTML code that produces the "Back to Campus" announcement above looks like this:

```
<img src="/sites/default/files/back-to-campus-banner.jpg" /><p style
="font-size: 18px;">As the university prepares for the start of the
fall 2021 semester, we remain #SouthernStrong, with <a href="https://
inside.southernct.edu/reopening#services">all of the following academic
offerings and campus services</a> in place for our students in a safe,
engaging campus environment. Our goal is to get you off to a great
start, whether you're returning to campus or joining our community for
the first time!</p>
```

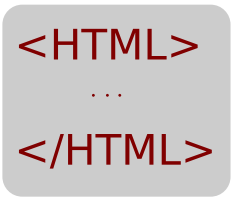
It is just simple text! No bells, no whistles. Parts enclosed by `<` and `>` are the markup. These parts will not be displayed themselves but rather describe how something is to be displayed and leaves the rest up to the browser. For example,

```
<img src="/sites/default/files/back-to-campus-banner.jpg"/>}
```

tells the browswer to insert an image (`img`) and tells the browser where to get the image (`src`). This part of the code creates

The rest of the code creates the paragraph underneath this image. It starts with `<p style="font-size: 18px;" >` — which tells the browser to start a paragraph (`p`) using an 18 pixel font—and ends with `</p>`, which tells the browser this is the end of the paragraph. In the middle of the paragraph is what makes webpages webpages—a hyperlink. The text between `<a href="https://inside.southernct.edu/reopening\#services" >` and `</a>` is marked up as a link, it's displayed in blue so the reader can see that these words are where they can click. Clicking them will bring up a new webpage in the browser.

Lab: 1
Working with markup



To see markup in action, point your browser to

https://htmledit.squarefree.com/,

where you can write some hypertext markup and see how it looks on your browser. The blue box at the top holds the markup, and it can be edited by you! The box below shows how the browser renders the markup. Do the following exercises on the `squarefree` webpage and answer the questions.

1. Insert `<em>` before the word magically and insert `</em>` after the word magically. What did this accomplish? Note: `em` is short for *em*phasis!

2. Copy and paste the following code into the webpage.

```
<p>A list of some common HTML markup
<ol>
 <li><tt>p</tt> is short for <b>p</b>aragraph</li>
 <li><tt>a</tt> is short for <b>a</b>nchor (which can
indicate a link or a place to link to)</li>
 <li><tt>ol</tt> is short for <b>o</b>rdered <b>l</b>ist</li>
 <li><tt>li</tt> is short for <b>l</b>ist <b>i</b>tem</li>
</ol>

Notice that markup can be nested -- the b and /b tags above are
inside the li and /li tags, which are between the ol and /ol
tags.</p>
```

3. What happens to text between the `<b>` and `</b>` tags?

4. What happens to text between the `<tt>` and `</tt>` tags?

5. Now change the `<ol>` and `</ol>` tags to `<ul>` and `</ul>` tags. What happens to the displayed page (in the white box)? Note: `ul` is short for **u**nordered **l**ist.

6. Notice how the b and /b tags in the last sentence are missing the usual angle brackets? What happens if you put them in?

7. This is a problem in all markup languages – some characters have special meaning. In HTML there are so-called escape characters to work around this issue. Google the phrase "html escape characters" and see if you can re-write the last sentence so that things **look like** a tag, but don't **act like** a tag!

2.2 LaTeX

Donald Knuth is an amazing Zen master of Computer Science and Mathematics. He created a program called TEX (pronounced like "tech") for typesetting – especially typesetting with mathematical content. Knuth wrote TEX mainly as an example of a style of coding that he called "literate programming." Essentially this means that the programmer creates the documentation for their work at the same time they write the actual code. Whether it's because of it being written in the "literate programming" style or simply because of Knuth's amazingness, TEX is renowned as one of the most bug-free systems in existance.

TEX, and a follow-on program known as LATEX, which builds on TEX and extends it, have become the standard(s) for communicating mathematics. We'll be talking only about LATEX from here onward. LATEX is much like HTML – it is a sort of markup language that gets processed and turned into a document suitable for visual display. The appearance of LATEX source code is very different from that of HTML source code, but in many ways they are incredibly similar! There are two marked distinctions: HTML is meant to describe how to lay out a page so that the content is clear, and do so on many different displays – the web-browser is empowered to alter things to aid in that clarity. On the other hand, LATEX is much more focussed on layout – the first thing one does in a LATEX document is to specify the page size. The author of a LATEX document can closely control where things will end up on that page[1]. The other big difference between these systems is how well they handle mathematical content. LATEX is very, very good at rendering mathematics and mathish notation from other fields like Chemistry. HTML stinks at that, which is completely weird since the original HTML standard was created by a physicist at CERN, and physicists tend to use a lot of math...

In a nutshell, HTML (the markup language of webpages) is used to control how a webpage appears. LATEX plays the same role for printed documents like books, reports, letters, or labs (like this one!). LATEX is a markup language that excels at creating technical documents, especially those that include mathematical formulas. You might want to read this blog all about why you want to learn LATEX.

Like HTML, LATEX uses tags to mark how text should look, to insert graphics, to create lists, and so on. Markup in LATEX begins with a \ (a backslash) as in `\pi`, which would be used to render our friend, π. Every LATEX document begins with the `\documentclass` tag, which specifies what type of document is to be created (book, article, report, etc.). This is similar to HTML documents, which all begin with the `<html>` tag and end with

[1] There is a strong argument against doing so – the author should really concentrate on content, and let the software control the individual pixels on the page.

the `</html>` tag. After the documentclass tag a LaTeX document must include the tags `\begin{document}` and `\end{document}`.[2] As you can probably guess, these tags mark the beginning and end of the content of the document. One of the simplest LaTeX documents possible is this one:

```
\documentclass{article}
\begin{document}

Hello World!

\end{document}
```

Just like HTML code, it isn't pretty! However, it tells the LaTeX renderer what to do. This code will create an "article" with the sentence "Hello World!" in it.

Let's give it a try!

The next lab will lead you through the process of creating LaTeX documents on a web-based service known as Overleaf.

[2] This is similar to HTML documents, in which the beginning of the content is marked by `<body>` and the end by `</body>`.

Lab: 2
Getting started with LaTeX

1. Go to https://www.overleaf.com/register
2. Register for a free account (probably best to use your university email.) You'll have to create a password at this point - make a note of it.[3]
3. It's probably best to skip their "Try the premium version for free" offer.
4. Click the "Create First Project" button – choose a blank project and name it "hello."
5. You'll see two main panels (there's also some junk above and to the left, but ignore all that for now.) The left-hand panel contains the LaTeX source code for your project and the right-hand panel gives a preview of the resulting document.
6. The "blank project" isn't completely blank. The source code panel will be pre-populated with:

```
\documentclass{article}
\usepackage{graphicx} % Required for inserting images

\title{hello}
\author{myemail}
\date{August 2023}

\begin{document}

\maketitle

\section{Introduction}

\end{document}
```

[3] Overleaf isn't exactly a "high stakes" setting, your password needn't be super complicated – just don't re-use a password that protects a more critical account!

You should see your document on the right side. It has a title section (which is what the `\maketitle` command created) and a section heading (this is what the `\section{Introduction}` command did). Scroll down to the bottom of the page and you will see the number 1, the page number. Pages of articles are normally numbered, so LaTeX puts that in for you! The area between the `\documentclass{article}` and the `\begin{document}` tags is known as the **preamble** of the LaTeX document. You should see that the preamble contains some commands that effect how the title looks. The default stuff that Overleaf stuck in there probably isn't quite what you want.

Let's fix that!

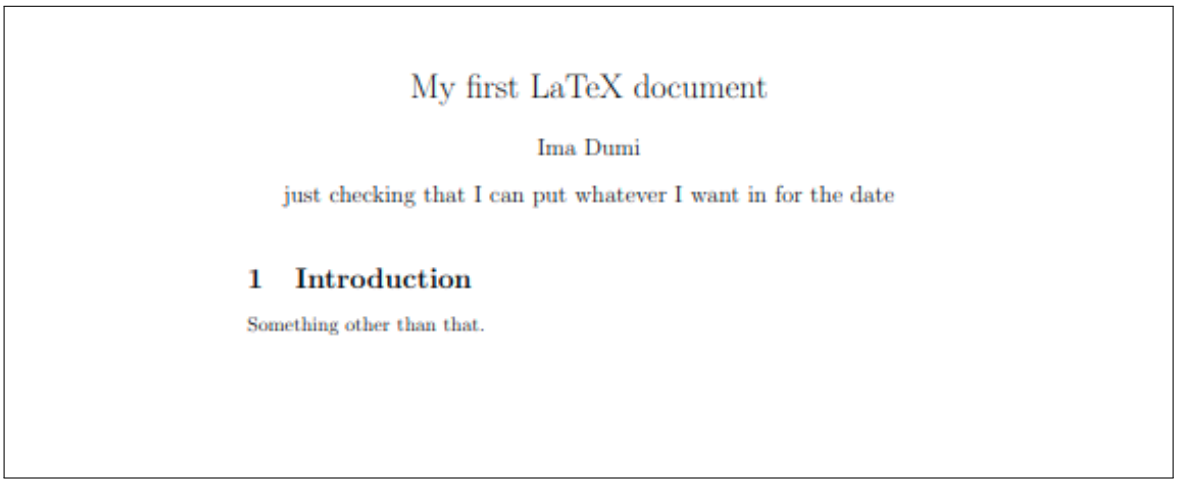
7. Make whatever changes you deem appropriate to the title, author and date commands. (BTW, "date" doesn't necessarily have to literally be the date.) To see what the effect of your changes is, you'll need to press the "Recompile" button.

8. Put some words, introducing yourself after the `\section{Introduction}` command, and recompile. (There is an old and slightly unfunny tradition that the first program you write when learning a new programming language is called "hello world!" Please make your introduction something other than that.)

At this point the source code might look something like:

```
\documentclass{article}
\usepackage{graphicx} % Required for inserting images

\title{My first LaTeX document}
\author{Ima Dumi}
\date{just checking that I can put whatever I want in the date}

\begin{document}

\maketitle

\section{Introduction}

Something other than that.

\end{document}
```

Which should render like so:



Adding a list

Now add a list of your three favorite classes of all time—of course math class is first on your list, so that one has been put in for you. You'll have to supply the next two...

1. Copy and paste the following markup into your document after your greeting, but before `\end{document}`.

```
\par
My three favorite classes of all time are

\begin{enumerate}
  \item Math
\end{enumerate}
```

   Notice that the `\par` tag does not have a begin or end. It only marks where a new paragraph should start. Same with the `\item` tag. It only marks where a new item in the list should start.

2. Add two items to the enumeration (and you can change the first item if by some strange chance math is not your all time favorite class).

3. There are several list-making environments in LaTeX. Try some googling (Maybe "list making latex environments") and you should discover the other list-making environments.

4. Make versions of your list of favorite classes that are (1) bulleted rather than numbered, (2) name the class and also give a comment about ~~why math is so awesome~~ why you like it.

Adding an equation

Equations in LaTeX come in two varieties—inline and display. An inline equation is any mathematical expression that appears in the middle of a sentence (like the π right here and earlier in this document). A display equation is any mathematical expression that should appear centered on its own line (because it's super important or just because it's too big to put in the middle of a sentence).

To put an equation in the middle of a sentence, enclose the math between two dollar signs ($). To add a display equation, enclose the math between double dollar signs ($$). Try it!

1. Copy and paste the following markup into your document.

```
My favorite mathematical constant is $\pi$, but I like $e$ too.
Did you know that $$e^{i\pi}=-1?$$ Weird...
```

2. Notice that exponents are typeset using the same notation as used on a calculator! Can you add markup to your document that will produce the following?

> The Pythagorean Theorem states that if a triangle has legs of lengths a and b and hypotenuse of length c, then
>
> $$a^2 + b^2 = c^2.$$

2.3 More LaTeX

Before we get to working a bit more with LaTeX, let's do some configuring that will save you a little typing going forward. When you create a new project in Overleaf, the system automatically puts your name in as the document's author. Unfortunately, it usually doesn't get your name right. Click the "home" icon (in the upper left, next to "Menu"). There is a drop-down labelled "Account" select "Account Settings" from it and then fix your first and last names.

Now if you create a new blank project, the preamble of your new documents should look something like this:

```
\documentclass{article}
\usepackage{graphicx} % Required for inserting images

\title{hello}
\author{Joe Fields}
\date{August 2023}
```

Notice that commands all start with a backslash followed by the command name, followed by something in curly braces. Well, not quite – the thing in curly braces (a.k.a french braces) is called the argument of the command – some commands don't have arguments, others have more than one. So the very first thing in the preamble is a `documentclass` command, whose argument is the word "article." Often such a command will also have optional arguements – these are collected in square braces between the command name and the curly brace arguments.

For example:

```
\documentclass[12pt]{article}
```

Would make an article with a slightly larger font. (Font styles and size can be set inside the document too.) There are quite a few other optional arguments available.

The second line of the preamble has a couple of things worth mentioning. The first is that there is a comment. Any text after a % sign is ignored, so this gives you a way to write little "note-to-self"s. The second is what the line is doing – adding to the default behavior of latex. There is a large community of LaTeX users who actively develop new features. These features are made available through packages.

The LaTeX source for this book makes use of a lot of packages: hyperref, geometry,

fncychap, babel, makeidx, graphicx, rotating, amssymb, amsmath, amsthm, color, and ulem. We'll explore a couple of those in the lab.

Lab: 3
More with LaTeX

1. Did you fix your name in the Overleaf "Account" area?
2. Okay. Then, start a new blank project called "more"
3. Edit the first line of your new project to include an optional argument for the `\documentclass` command:

```
\documentclass[12pt]{article}
```

4. Also, add the following lines into the preamble of your document:

```
%\usepackage{geometry}
%\usepackage{hyperref}
```

   Of course, the commands you've just entered start with the comment symbol so they won't do anything. Did you notice how the Overleaf source editor renders them in green?

5. Now, add the following to the body of your document. (After the `\section{Introduction}` command)

```
The first thing you should notice is that the text is a little
bigger than it was last time.  You may also have noticed that
the margins are a bit big by default.  For illustration purposes
we need a paragraph that is long enough to ensure we've
reached the right margin.  This one should do the trick.
```

6. Hit the "Recompile" button. Did you notice that the text is a little bigger than it was last time?

7. Try deleting the `[12pt]` optional argument to the `\documentclass` command. Hit "Recompile." Do you see the difference now?

8. Put the `[12pt]` optional argument back and recompile one last time.

9. Okay, now what about that comment about LaTeX's margins? Try uncommenting the line that includes the `geometry` package. Recompile. What's changed?

10. The `geometry` package will give you minute control over your document's margins. The default margins when the `geometry` package is in use are a good bit smaller than LaTeX's defaults.

11. Try replacing the `geometry` line with this one:

```
\usepackage[bottom=1in, right=.5in, left=.5in, top=1in]{geometry}
```

12. Play around a bit. What settings for the margins are most pleasing to your eyes?

13. Now, uncomment the `\usepackage{hyperref}` command. Recompile.

14. Did you see anything change?

15. If you did, it's probably your imagination. Hyperref doesn't make any visible changes – it provide you with some new commands!

16. Drop the following line into your source file:

```
This is a
\href{https://www.cespedes.org/blog/85/}{link}
to a page that gives some good information about
special characters in \LaTeX{}.  You may notice
that the {\tt href} command provided by the
{\tt hyperref} package is our first example of a
\LaTeX{} command that has two arguments.
```

17. I don't know about you, but I really don't like how hyperlinks are rendered by default. If you put the following right after the `\usepackage{hyperref}` command you'll get a nicer appearance.

```
\hypersetup{colorlinks=true,  urlcolor=blue}
```

2.4 even more LaTeX

There's a term you'll hear in the publishing world and sometimes in Computer Science: whitespace. It's a collective term for all of the symbols that don't end up putting ink on the page. Basically, the whitespace characters are: the space key, the tab key, and the Enter key. In LaTeX the only thing whitespace characters do is seperate the printable things (ordinary characters and LaTeX commands). In particular, they don't add extra space! You can put a single blank line between two paragraph, or you can put 27 blank lines. The output will look the same. THIS CAN BE MADDENING! You notice that an inline equation looks a little pinched in too close to the surrounding text, so you put in some extra spaces, and it has no effect. So you put in some more spaces, and still nothing happens. They say insanity consists of repeating the same experiment expecting different results. Try not to travel too far down that path! Maybe repeat the mantra "whitespace is just whitespace" while you're doing the next lab – don't expect it to produce actual space...

So, how do we produce actual space? How do we fine tune the space between things? (This might mean adding or subtracting space.)

First, you should probably ask yourself if the change you want is really desireable. LaTeX has some pretty good algorithms for calculating spacing...

Second, you should check if there's a systematic way to make the change you want. For instance, if you want a little additional space between your paragraphs, you can put something in the preamble that adjusts those inter-paragraph spaces throughout the document:

```
\setlength{\parskip}{8pt}
```

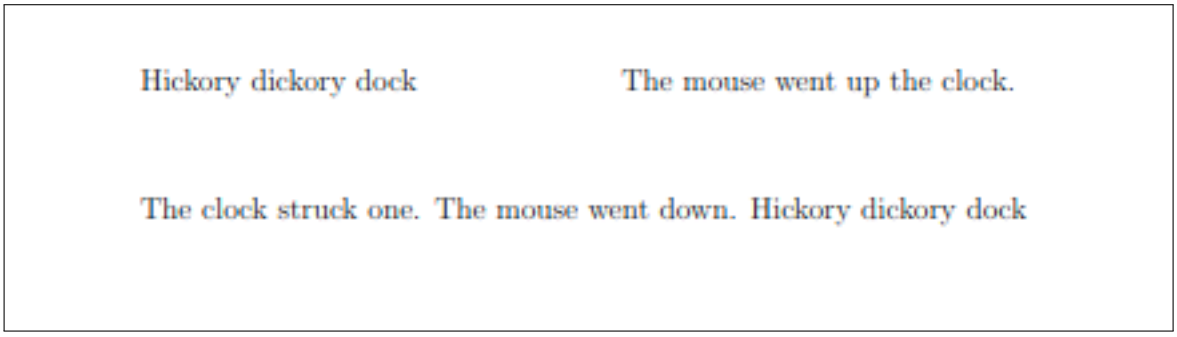
or, if you want your whole document to be double-spaced:

```
\linespread{2.0}
```

But, if you really want fine-grained control there are commands for adding vertical and horizontal space:

```
Hickory dickory dock \hspace{1in} The mouse went up the clock.

      % There should be blank lines surrounding the space command
      % Cutting and pasting might leave them out, so add them
      % back in if they're missing.

\vspace{.5in}

The clock struck one. The mouse went down. Hickory dickory dock
```

Will give output that looks like this:



In math mode there are special spacing commands (`\,` `\;` `\!` `\quad`) that we can use to make finicky adjustments to the positions of symbols.

If all else fails there are struts. LaTeX has a command that's meant to create a rectangular blob of ink – `\rule{width}{height}`, but if you make one of those lengths 0 the resulting blob won't have any ink in it – it'll be an invisible thing that just takes up space – that's called a strut. I sincerely hope you don't find yourself creating struts in this course. They're really frowned upon – there's almost always some better way of handling things than making an invisible space-taking-upping thing. Just thought you should know, 'cause sometimes...

Lab: 4
More more with LaTeX

1. Go to Overleaf and, instead of a blank project, under New Project select "Example Project."
2. Take a little time to go through the source code line-by-line and figure out what things do.
3. Try the following matching exercise.

`\href{https://www.overleaf.com/learn}{help library}`	A. Set the author field
`\author{You}`	B. A hyperlink
`\includegraphics[width=0.25\linewidth]{frog.jpg}`	C. Start a numbered list
`\url{https://www.overleaf.com/contact}`	D. Some displayed math
`\usepackage[english]{babel}`	E. Puts in an image
`\begin{enumerate}`	F. Sets the document language
`\[S_n = \frac{X_1 + X_2 + \cdots + X_n}{n}` `= \frac{1}{n}\sum_{i}^{n} X_i\]`	G. An alternative form of a hyperlink

4. You may have notice the use of `\[` and `\]` to begin and end the display math environment. Using `$$` to begin and end a displayed math environment (or `$` to begin and end an inline math environment) has an unfortunate consequence. Users have a tendency to forget either the begin or the end tag which leads to errors at compile time that are really hard to figure out. If you get compilation errors that seem plain crazy, check to see if every 'begin math' token has a matching 'end math' token. The problem is a little easier to deal with when the begin and end tags are different.

   The begin/end version of the tags for displayed math are `\[` and `\]`.

   The begin/end version of the tags for inline math are `\(` and `\)`.

   So, either use these – or take my advice and learn to scan your code for mismatched dollar signs!

   Next we're going to explore more of the features of the Overleaf interface. The panel to the left of the 'Edit' panel contains two subparts, above is a file listing. The current project has 3 files in it. The sub-panel below shows the logical structure of the document - you can click on things to move to the spot in the source code where they are defined. The panel that runs along the top of the page contains the Overleaf menu and the Home button on the left and towards the right we see Review, Share, Submit, History and Chat. Let's start with Share.

   ---

5. Find a partner, one of you create a new project (use the Example Project template again) then share it with the other. If you both go to the Home area in your respective instances of Overleaf, you should each be able to see the project. You should be able to edit the project simultaneously!

6. One of you should adjust the `\author{}` command so that it contains both your names. The other should add a comment to the file, maybe "I think my name should be first." To add a comment, click the Review button which opens a new panel between the source and the pdf panels – highlight some text in the edit panel and you'll be prompted for your comment.

7. The Overleaf interface contains elements for resizing and hiding the various panels. Play around with customizing the interface. (For instance I often work with only the Edit and Review panels open - then I just need to unhide the Pdf panel when it's time to recompile and see how my changes look.)

8. When working collaboratively, but not physically near one another, the Chat panel can be quite useful. Send each other a chat!

9. One last thing. Did you notice how the size of the little frog image was specified? You can also use an actual length, but the strategy used here (making the image's width be a percentage of the width of a line) is smart. Other possibilities are to give a scale factor:

```
\begin{figure}
\centering
\includegraphics[scale=.25]{frog.jpg}
\caption{\label{fig:frog}This frog was uploaded via the file-tree menu.}
\end{figure}
```

   Make that change and recompile.

10. You should notice that the frog picture is somewhat bigger, and also that it appears in a different place! That's because things like tables and figures are known as "floats." The typesetter (in this case the LaTeX software) has considerable latitude about where they end up in the layout. Because it's possible that a table or figure ends up far from where it's referred to in the text, these "floats" are usually numbered. As you work and revise a document, it's quite possible for new diagrams and tables to move things about and change those numbers. Look for the commands `\ref{}` and `\label{}` in the source code to see how LaTeX can automatically keep these number references current (with a little help from the author).

11. Notice that in one part of the current project there is an example of a simple table. Create a new section, and add a table to your document which tabulates some data you get from 3 or 4 classmates: name, favorite animal, favorite color, favorite automobile – actually, pick your own favorite somethings! Use the "table" environment as a container for your table (which should be created using the "tabular" environment) make sure that the table has an appropriate caption and that there is a `\label{}` command. The typesetter may decide to put your table somewhere else in the document! Write a short paragraph about all this stuff and include a `\ref{}` to your table.

2.5 MathJax

LaTeX is getting somewhat dated. It seems likely that it'll be supplanted by something more user-friendly at some time during your professional career. Still, the notation that LaTeX uses to encode mathematical content works incredibly well. It pops up in unexpected places. For instance, when Mathematicians and/or Scientists are communicating via email, and some math needs to be expressed, out come the dollar signs! The object of study in this section is MathJax – a javascript package that allows one to include mathematical content into webpages. Perhaps it won't be surprising to learn that LaTeX math notation is involved!

Unfortunately, the sandbox where we played with HTML markup previously doesn't allow for scripting (and MathJax is a script) so we will need to work in a more permissive environment. Google makes a free service called Sites available to the public. This is "free as in beer" not "free as in liberty" but still, it's pretty cool that they make something available to the general public that lets folks create websites in a (more or less) "What You See Is What You Get" environment. But, there is a bit of a backdoor! One can also drop arbitrary HTML code into a Google sites page.

That's where we are going in the next lab.

Lab: 5
MathJax

1. To do this lab, you'll need a free Google account. If you don't have one and are unwilling to sign up for one, your instructor may be able to provide you with login credentials for a dummy account.

2. Verify that you are signed in to your Google account (for instance check if you can see your "Gmail" email.)

3. Point your browser to `sites.google.com` and start a new, blank, site.

4. On the right-hand side of the screen there is a menu system containing a variety of elements you can add to your site. Select the one that says "Embed." In the dialog that pops up, switch it over to "embed code" and paste the following into the text box:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>MathJax Example</title>
    <script id="MathJax-script" async
          src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-chtml.js">
    </script>
</head>
<body>
<H2> Testing MathJax </h2>
<p>
If we are given a quadratic polynomial \(Ax^2+Bx+C\), we can find its zeros
using the quadratic formula, a relatively simple expression in terms of the
coefficients \(A\), \(B\) and \(C\) which gives the \(x\) values that satisfy
\(Ax^2+Bx+C = 0\).
<p>
The formula is
\[ \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}. \]
</body>
</html>
```

5. If you want to make changes to your embedded element, click somewhere in it (to select it) and notice the icons that appear in the upper-left corner. Select the pencil to edit. Also, the text area that contains your code is very tiny, but it has a resizing handle in the bottom right corner.

6. Turn your attention to the tool area at the top of the window towards the right. The first three icons there are "undo," "redo," and "preview." It's probably obvious that undo and redo are useful! The preview button is pretty cool too, it let's you see how your site will appear on a regular computer, or on a tablet or cellphone. The blue X leaves preview mode.

7. We've basically just hijacked Google sites to create a sandbox for playing around with LaTeX math! A lot of people prefer to use the $...$ notation for math. You can enable that by adding the following magic spell to your preamble:

```
<script>
  window.MathJax = {
      tex: {
          inlineMath: [['$', '$'], ['\\(', '\\)']]
      }
  };
</script>
```

8. Try the following. Count on your fingers but doubling with each step instead of adding 1 with each step.

   (a) 2

   (b) 4

   (c) 8

   (d) 16

   et cetera

   Of course, repeated multiplication is the same thing as exponentiation, so if you made it through all ten of your fingers, you should know the value of 2^{10}. You could also work backwards (cutting in half with each step) to figure out the value of 2^0. Or you could enlist a friend and work out what 2^{20} is.

   Create a little webpage - using MathJax for the math - explaining all this for someone who's in, say, Middle School (maybe 6th or 7th grade). See if you can use the intuition from this physical activity to explain that weird rule about powers: $a^b \cdot a^c = a^{b+c}$.

2.6 Project

The last item in the previous lab had you getting started on the project.

Make a website using Google Sites that explains the laws of exponents to a middle school student. You should include something to motivate the students as to why we would want to look at powers this closely. Repeated doubling (as in the lab) is just one example of an exponential process, but it yields lots of stories that could serve as motivation.

- The fact that you can't fold a sheet of paper in half more than 7 times.
- The paradox that the number of your ancestors as you go back in time will eventually be greater than the entire human population!
- The way a disease spreads through a population

Of course, beyond motivation, you'll need to actually explain (for example) why $b^x \cdot b^y = b^{x+y}$ and similar facts about exponentiation. For that you'll need some math notation, and for that you should use MathJax!

3 Symbolic Computer Algebra

3.1 Computations using computer algebra

A *computer algebra system* (or *CAS*, for short) is software that is able to perform algebraic procedures similar to what you would normally do "by hand." There are several popular CAS used in mathematics, with the most common being Mathematica, Sage, and Maple. We will be using Sage, as it is available as free open-source software, and it has been largely developed by mathematicians, integrating many specialized packages used by casual and professional mathematics researchers. Much of it uses Python, so if you have any knowledge of coding in Python, you will find Sage to be familiar; but fear not if you do not have any prior experience, as the purpose of this lesson is to get the hang of the basics!

Sage can be installed on your computer or run from the CoCalc servers if you create a (free) account. For now, we will use a lightweight version of Sage that can be run on a webpage without any installation or other hassles. Go to https://sagecell.sagemath.org/.

You can enter basic Sage commands into the textbox on that page and click "Evaluate" to get the output. We will begin by testing out some basic arithmetic operations.

Lab: 6
Sage Cell Server

Go to https://sagecell.sagemath.org/.

You can enter basic Sage commands into the textbox on the page and click "Evaluate" to get the output. To get started let's use sage like a fancy calculator. The operands for sum, difference, multiplication, and division are `+, -, *, /` .

Try the following operations:

1. Try the following operations:
   (a) `1 + 2 + 3`
   (b) `42 - 23`
   (c) `1331 * 11`
   (d) `144 / 9`
2. That's pretty standard stuff, you could more easily do those on your phone... But, can your phone calculate 200 digits of π in the blink of an eye? Try this:

   ```
   n(pi, digits=200)
   ```

3. You may have noticed that the previous question's answers were all whole numbers, but this last one was a decimal. We've been holding out on you a bit – sage does exact computations. Unless you ask it for a numerical approximation (which is what the `n( )` function was all about) it will give answers that are 100% precise – but sometimes that doesn't seem very helpful!

   Try evaluating these:
   (a) `pi`
   (b) `sqrt(2)`
   (c) `7/3`
4. Sage and its brethren are Symbolic Computer Algebra Systems. The reason it just parrots back the question to you has to do with the "Symbolic" part of that. For

example, the `sqrt(2)` thing above is regarded by Sage as a symbolic entity. It is a full and precise representation of the number we write as $\sqrt{2}$. It isn't some lame 4 or 10 or 100 digit approximation of $\sqrt{2}$. It is the real deal. Let's see what happens when we raise `sqrt(2)` to various powers (you can use either `^` or `**` for exponentiation in sage).

Try evaluating these:

(a) `sqrt(2) ^ 2`

(b) `sqrt(2) ^ 4`

(c) `sqrt(2) ^ 7`

Was the answer to that last one suprising? Or does it make sense in retrospect?

5. Much of the power of Computer Algebra comes from the same feature that makes regular Algebra useful – variables.

   There are two kinds that we need to distinguish: computer variables and mathematical variables. Computer variables are pretty easy, you can just make up whatever name you want and then start using it. For example:

   ```
   myvar = sqrt(2) ^ 7
   n(myvar)
   ```

   BTW, notice how the default version of the `n()` function only gives about 13 decimal places? Try this too: `n(myvar, 100)` (that's what 100 bytes of accuracy looks like – if you want a specific number of decimal places, see the example involving π up above.)

6. Mathematical variables are handled a little differently. First (other than x which is there automatically) you have to declare them to the system. The syntax looks like this: `y = var('y')`. What's happening there is that we're telling the system that `y` is a variable, but also that we want the system to print 'y' when referring to it. I think you'll get the point if you figure out what happens when you evaluate `y = var('w') ; 6*y`.

7. Let's try a little mathy something - with a couple of variables. Remember how the equation of a line is usually written as $y = mx + b$? Put the following into the sage cell:

```
y=var('y')
m = 2 ; b = 1
y=mx+b
```

The error message you get when evaluating this should look like:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In [1], line 3
      1 y=var('y')
      2 m = Integer(2) ; b = Integer(1)
----> 3 y=mx+b

NameError: name 'mx' is not defined
```

To be fair, Sage usually puts out error messages that are on the cryptic side. But not this time! The message says "NameError" and it's even highlighted the thing that's wrong.

If you fix the error[1] something else a little unexpected happens. Nothing! There's no output...

This is just because the last line is an assignment – it has no return value. A very common "idiom" you'll see when looking at other people's sage code looks like this:

```
y=var('y')
m = 2 ; b = 1
y=m*x+b ; y
```

The semicolon is just a way to sneak multiple commands onto a single line. The difference here (as opposed to the code above) is that the last command isn't the assignment to the variable `y`, it's just `y` by itself - and that does have a return value, namely the contents of the variable `y`.

8. Sage has a function called `sum()` that adds things up. Next time we'll learn how use the help facility to find out how to use a Sage command, but for now, we're just going to tell you: the `sum()` function takes four arguments. The first argument is a formula for the terms of the sum, the second argument is the so-called summation variable, the last two arguments tell where the sum starts and ends.

   For example

[1] A takeaway from this is that you can't rely on implicit multiplication. If you write `7x` the system is going to think you've created a new (computer) variable whose name is "7x."

```
k=var('k')
sum(2*k+1, k, 0, 5)
```

would give us the sum of all the odd numbers from 1 to 11.

What is the sum of all the odd numbers from 1 to 201?

9. Use Sage to calculate 2^0, $2^0 + 2^1$, $2^0 + 2^1 + 2^2$, and $2^0 + 2^1 + 2^2 + 2^3$. Continue adding the next largest power of 2 until you notice a pattern in the result. What is the pattern?

10. The Sage summation facility can also do things entirely symbolically. This is sort of like the difference between computing a particular value and giving a general formula. Try the following in the Sage cell and see if you and Sage found the same pattern.

```
k, n = var('k, n')
sum(2^k, k, 0, n)
```

3.2 Sage on CoCalc – the interface

For more advanced calculations that require multiple steps, it will be beneficial to signup for CoCalc. CoCalc is a cloud-based service[2] that gives its users access to many software packages. Originally it was Sage only, but now, in addition to Sage, one can run Python notebooks, LaTeX documents, get a full Linux terminal, run a so-called "computational whiteboard," even manage a course on CoCalc. The CoCalc interface allows for sharing projects and working on thing collaboratively with one's partners – edits become visible to the other party in real time.

Because of its genesis as a web interface to a free, open-source project, CoCalc makes free accounts available. One can also purchase "upgrades" that give certain perks (most notably, access to less highly-loaded servers for your computations). The free accounts come equipped with an "annoyatron" banner that urges you to get a payed account. Many Colleges/Universities will be able to provide you with upgrade tokens so you will have the more premium experience.

This lab will lead you through the process of obtaining a CoCalc account and getting started with the Sage notebook interface.

[2] "In the cloud" really just means "on somebody else's computer." There are a ton of servers out there that can be rented on flexible terms, CoCalc just purchases computing power as needed...

Lab: 7
Sage on CoCalc

1. Go to https://cocalc.com/ in a web browser.
2. At the top right of the screen, click on the "Sign Up" link.
3. You'll need to check the box that says "I agree to the Terms of Service." There is also a "Privacy Policy" that is included by reference into the Terms of Service.
4. It also asks you to select the software that you plan to use. At a minimum select "SageMath." There's also an "Everything!" option if you feel adventurous.
5. Next you'll be asked for your email address, your name, and to choose a password – please make it a strong one!
6. You should now see a page that says "Signed in as XXXX XXXX" at the top of the page.
7. Click on the "Projects" link at the top left of the page.
8. In the textbox that says, "Project title – you can easily change this at any time!" enter a name for your new project, then click on the "Create New Project" button.
9. On the next page, click the "New" link near the top of the page. This is going to let you create a file inside your project.
10. Pick a name for your file, select "Sage worksheet," then click on the "Start project" button at the top of the page.
11. Unlike in an embedded SageMathCell on a webpage, CoCalc will provide output for multiple calculations at once, and it will allow you to keep previous calculations on the screen so that you can change them or refer back to them.
12. The interface you're looking at is called a Sage notebook. A notebook consists of an alternating sequence of input and output cells. There's a thin line between an input cell and the corresponding output cell, and the output cells have a big green border on the left side. Both sorts of cells can be shown or hidden using the little triangles in the left margin. When you first start up a fresh notebook, there is only the first

input cell (with a line numbered 1 in it). Try adding `1+1` in that cell. Watch for the pulsing green "busy" signal[3] and take note of how the interface changes when you execute the cell. (To execute, you can either hit the "Run" button or hold down the shift key while hitting the Enter key on your keyboard.)

13. If you hover your mouse over the line before an input cell it will turn blue. If you then click it you'll get a new cell. It's usually a good idea to break tough computations up into manageable chunks. That's what cells are for. Just be aware that the cells in a notebook aren't independent – if you set a variable to some value in one cell, the other cells will know about that. Create a new cell above your `1+1` cell and ask Sage to multiply a couple of random number with 10 or so digits each. Take note of how long the "busy" signal lasts this time.

14. Let's re-emphasize a quirky thing about Sage – assigning a value to a variable doesn't create any output. So if you want to have the computer verify that it really did what you just told it to do (i.e. print some output) you need to write the variable a second time. This often has the form of some long computation followed by a semicolon and the name of a variable. Try pasting the following into a new cell and hitting Shift+Enter.

```
A, P, r, n, t = var('A, P, r, n, t')
P=1000
r=.06
n=12
t=5
A = P * (1 + r/n)^(t*n) ; A
```

15. Another point that bears repeating: The instructions in a cell need to be executed in order to have any effect. It's not enough to put your cursor in the cell and push "Enter" (that just adds a linebreak, making the cell longer). You have to hold the "Shift" key while pressing "Enter" if you're trying to tell the computer to "Do it!" (You can also select the cell and click the "Run" button in the toolbar, but you'll soon find that Shift+Enter is a lot quicker.)

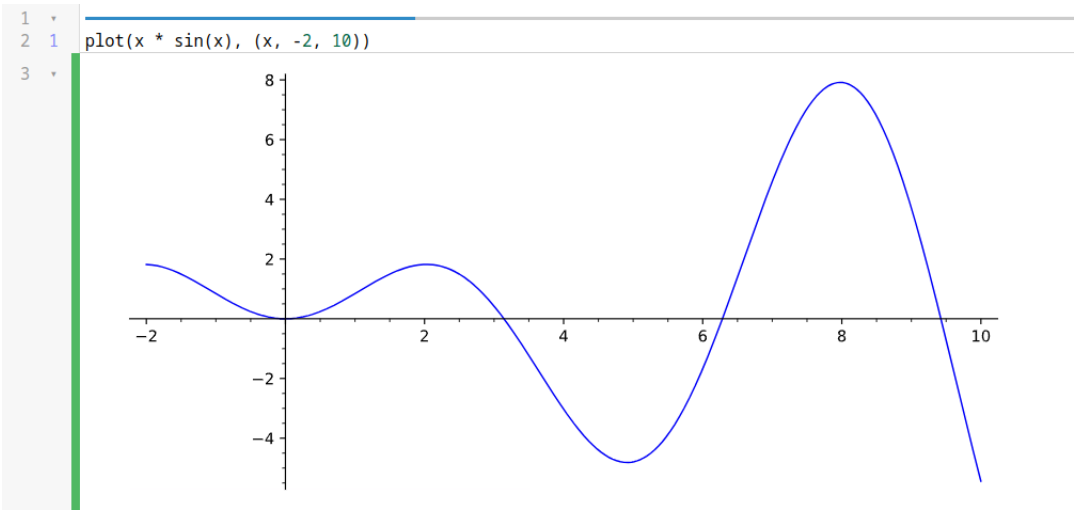
[3] While sage is "thinking" the interface provides a visual indication. A lot of the delay is about CoCalc obtaining and setting up a fresh cloud server to run your project on, so after the first computation things should go a lot faster.

16. There are two very useful ways of getting help within sage: tab completion and help messages. Tab completion is something you may have encountered in other computer applications. If you type the first few letters of a command and then hit the "Tab" key, the machine will either complete the typing for you, or give you a list of possible completions. Try typing "plo" in the next cell and then hitting "Tab." You should see that Sage has a `plot()` function, but also it can do 3-dimensional plots – and there are many other arcane-looking plots commands!

17. Ok, you just discovered that sage has a `plot` command – but how do you use it? For that, it's often a good idea to look at the help message for the command. To get help, just type a question mark after the name of the command (then "Shift+Enter").

    I usually just scroll down through the help message until I find some examples...

18. Speaking of examples, the tool bar above the cells is full of them! Create a fresh cell, then find the "Plots" drop-down and select "Function" Hit Shift+Enter and you should see something like:



19. Even more help is accessible from the top menu entry that's labelled, uhmmm, "Help." There you can access the sage documentation, submit a support ticket if you think you've found a bug, or even have a chat with ChatGPT. Try it!

20. When executing a cell produces an error message you can even ask for "Artificial Intelligence help" in correcting the problem. It's possible to hold a back-and-forth conversation with ChatGPT that feels almost like talking to a human!

21. Some of you have probably already studied Calculus. For others it will be coming up soon! The two main operations in Calculus (which are in a certain sense inverses of

one another) are known as integration and differentiation. Try using tab-completion and the help facility to find out how to do these operations in Sage. It's pretty likely that you'll do something incorrect and get an error message. If that happens try the "Ask ChatGPT" option. Turn it into a conversation where you go back and forth with the AI a few times!

22. Let's put these calculus operations to use! Use sage to find the integral and the derivative of the function $f(x) = x^2$.

3.3 Sage on CoCalc II – Algebra

The last lab mostly concentrated on getting you used to the interface. Of course, learning that interface would be pointless if Sage wasn't capable of doing something useful for us!

And it definitely can!

The next lab has us exploring Sage's abilities in Algebra. Recall the operations with polynomials that you learned to do by hand in Algebra class:

- Multiplying 2 polynomials (do you recall the FOIL mnemonic?)
- Factoring a polynomial (sort of the opposite of the previous thing)
- Evaluating a polynomial at a particular value of its variable.
- Solving – what x values give a particular output?

Not to say that that was wasted effort (if nothing else, your brains got stronger from the exercise!) but, yeah, Sage can do all of that and more...

Before we jump into the actual lab, a quick word about "equals."

There's really just one version of the "equality" idea in Mathematics. In Computing, there are two:

1. One place an equals sign shows up is when we're assigning a value to a variable.
2. The other place an equals sign shows up is when we're checking whether to variables actually contain the same content.

In Sage (and Python) the "assignment" version uses a single "=" – the "equality testing" sense use a double "==."

If you were to execute

```
x=1
x==0
```

in a Sage cell, the first line (being an assignment) will produce no output, and the second line will produce output informing us that no - one and zero are not the same.

Lab: 8
Sage on CoCalc II – Algebra

Most of the computations we did in the previous section can be done on a regular calculator, but where a CAS like Sage really shines is when working with algebraic expressions including variables. When working with Sage, we will need to declare symbolic variables before we use them.

```
x = var('x')
```

This tells Sage that we will be using the symbol/letter "x" as an unbound variable in an expression. We wish to treat it as symbol that can represent any possible value and not as a specific value that is fixed. In other words, x is a mathematical variable, not a computer variable.

```
x = var('x')
3*x+7*x+5
```

This represents the expression $3x + 7x + 5$. Note that the implied multiplication between 3 and x needs to be specified when typing into Sage.

Inputting

```
var('x')
3x
```

will just cause a NameError.

Sage can be helpful when working with algebraic expressions, as it can do things like expand, factor, or simplify expressions.

If we want to factor an expression like $x^2 + 4x + 3$, we can use the `factor` command.

```
x=var('x')
factor(x^2+4*x+3)
```

We can also expand out an expression using the `expand` command.

```
x=var('x')
expand( (x+1)*(x+3) )
```

It is also often useful to define a function, such as $f(x)$, much like we do in a regular math class.

```
x=var('x')
f(x)=x^2+4*x+3
```

We can then evaluate $f(x)$ at different values of x using the standard notation. For example, to evaluate at $x = 1$, we can then enter

```
f(1)
```

which will give us the output of

```
8
```

which is the same as the value of $(1)^2 + 4(1) + 3$, i.e. the value of substituting $x = 1$ into $f(x) = x^2 + 4x + 3$.

We can also easily apply the `expand` and `factor` commands to a function we have already defined.

```
f.factor()
```

will give the output

```
x |--> (x+3)*(x+1)
```

which is the same factorization as when we typed

```
factor(x^2+4*x+3)
```

Similarly,

```
f.expand()
```

can be used to `expand` the function $f(x)$. In addition, a function has the `full_simplify` option.

```
var('x')
f(x)=(x^2+4*x+3)/x+(x+3)*(x-1)/(x+1)
f.full_simplify()
```

Will simplify the very complicated expression of

$$\frac{x^2 + 4x + 3}{x} + \frac{(x + 3)(x - 1)}{x + 1}$$

into a single rational function.

Finally, sometimes Sage will give us an exact expression for something for which we would like a decimal approximation. For example,

```
var('x')
f(x)=(x^2+4*x+3)/x+(x+3)*(x-1)/(x+1)
f(2)
```

gives the output

```
55/6
```

Because Sage is mathematical software, and mathematicians usually want exact answers, that is what it will return, when possible. In order to force it to give a decimal expression, we can use the `n()` command.

```
n(55/6)
```

gives the decimal approximation of

```
9.16666666666667
```

Lab: 9
Finding Solutions in Sage

Solving equations

(Note: follow along with these initial computations then try the problems.)

We will investigate functions of the form $f(x) = ax^2 + bx + c$. To begin with, we will consider when $a = 1$, $b = 0$, and $c = 0$, i.e. $f(x) = x^2$.

Start by defining $f(x) = x^2$ in Sage.

```
f(x)=x^2
```

To solve the equation $f(x) = 1$ (i.e. $x^2 = 1$) in Sage, we can use the **solve** command:

```
solve(f(x)==1, x)
```

This will solve the equation $f(x) = 1$, solving for the variable x. Note that when solving, we need to put two equals signs to denote equality. This is because Sage will interpret a single equals sign as an assignment (i.e. we would be defining $f(x)$ to be the function that is always equal to 1).

The output we obtain is

```
[x == -1, x == 1]
```

These are the two solutions that we expect of $x = -1$ and $x = 1$.

Exercises

1. Try to solve the equation $f(x) = 4$ using Sage.
2. Solve the equation $f(x) = 3$ using Sage.
3. Solve the equation $f(x) = 0$ using Sage.
4. Do you expect for there to be any solutions for $f(x) = -9$? Try it in Sage and see what happens.
5. For what values of h does $f(x) = h$ have two real solutions? Only 1 solution? Two imaginary solutions?
6. Define a new function $g(x) = x^2 + 10x + 21$. Let's try two things:

(a) Use the `factor()` member function to see how $g(x)$ factors.

(b) Use the `solve()` command to see when $g(x)$ is equal to zero.

Can you explain the minus signs?

7. Do the things `factor()` and `solve()` in the previous problem for

(a) $g(x) = x^2 + 10x + 22$

(b) $g(x) = x^2 + 10x + 23$

(c) $g(x) = x^2 + 10x + 24$

(d) $g(x) = x^2 + 10x + 25$

(e) $g(x) = x^2 + 10x + 26$

Sometimes Sage factors the polynomial into linear factors.
Sometimes Sage is refusing to factor because the zeros are not nice numbers.
Once, Sage refuses to factor because things have gotten truly weird.

Identify which is which? What do you suppose `I - 5` means?

Graphing functions

(Again, follow along with the first few computations, then try the problems.)

Another useful way to analyze things is by visualizing functions using plots of their graphs. Let's begin by plotting the function $f(x) = x^2$:

```
f(x)=x^2
plot(f(x))
```

The first line defines the function $f(x) = x^2$, then the second line will plot the function. Note that since we did not specify the range for the axes, the plot will automatically pick a range. If we want to see more of the graph, we can try adjusting the **xmin**, **xmax**, **ymin**, and **ymax** values, which correspond to the minimum and maximum values of the x- and y- axes that we desire. For example, changing the plot command to

```
plot(f(x),xmin=-3,xmax=3,ymin=-2,ymax=9)
```

will gives us an x-axis that ranges from -3 to 3 and a y-axis that ranges from -2 to 9.

We can also plot multiple functions on the same plot, in case we want to compare them together. Let's try overlaying the graph of the function $g(x) = 1$ onto the same set of axes:

```
f(x)=x^2
g(x)=1
plot([f(x), g(x)],xmin=-3,xmax=3,ymin=-2,ymax=9)
```

We have defined two functions now, $f(x) = x^2$ and $g(x) = 1$. To plot both, we put them both into a list that is enclosed in square brackets, with the two functions separated by a comma. This produces graphs of both $f(x)$ and $g(x)$, in two different colors.

How many times do the graphs of $f(x)$ and $g(x)$ intersect?

Exercises

1. Graph the functions $f(x) = x^2$ and $g(x) = 4$ together on the same set of axes. How many times do the graphs of $f(x)$ and $g(x)$ intersect?

2. Repeat with $f(x) = x^2$ and $g(x) = 3$. How many times do the graphs of $f(x)$ and $g(x)$ intersect?

3. Repeat with $f(x) = x^2$ and $g(x) = 0$. How many times do the graphs of $f(x)$ and $g(x)$ intersect?

4. Repeat with $f(x) = x^2$ and $g(x) = -3$. How many times do the graphs of $f(x)$ and $g(x)$ intersect?

5. What is the relationship between the number of times that the graphs of $f(x) = x^2$ and $g(x) = k$ intersect and the number of solutions to $x^2 = k$?

6. Are there any intersections of the graphs $f(x) = x^2 + 1$ and $g(x) = x + 2$?
   Use a sage `plot()` command to visualize the situation, then use the `solve()` command to find the points of intersection.

7. Notice that in the last problem the `solve()` command only gives us the x coordinates of the points of intersection. How can we find the y coordinates?

8. Recall that the quadratic formula is

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

   What are the a, b and c in this case? Careful! You need to rearrange so that you have a quadratic polynomial set equal to zero.

   Does the answer from using Sage's `solve()` command agree with what the quadratic formula tells us?

Numerical Approximations to Solutions of Equations

(There are no problems in this section, so just read along and do the computations as you encounter them.)

Some equations are difficult to solve exactly even with the assistant of a computer and computer software, and Sage is no exception to this.

Use the **solve** function to have Sage solve the equation $x^5 - 3x^4 + x^3 + 2 = 0$.

The output that we get is

```
[0 == x^5 - 3*x^4 + x^3 + 2]
```

which is another way of Sage telling us that it could not find a solution. However, plotting the function $x^5 - 3x^4 + x^3 + 2$ tells us another story. Plot the graph of $x^5 - 3x^4 + x^3 + 2$ in Sage to see whether it has any roots. How many are there, and what are the approximate values from the graph? You may want to play around with the range of the x-axis (using xmin and xmax) to get a clearer picture.

(3 roots, roughly -0.8, 1.2, and 2.5)

We see that there are 3 roots, at roughly $x = -0.8$, 1.2, and 2.5. Although Sage cannot get the exact values for them by solving the equation $x^5 - 3x^4 + x^3 + 2 = 0$, it can get approximate decimal values for the roots by using the **find_root** function. In short, a computer algebra system such as Sage uses a sophisticated version of "find where the graph crosses the x-axis, and zoom in repeatedly around that point to get a more precise estimate of the x-coordinate of where the graph crosses the x-axis." To find a root x where $-2 < x < 0$, we would use the command

```
find_root(x^5-3*x^4+x^3+2==0, -2, 0)
```

The output of this is the root

```
-0.7931397744702121
```

If we want to find a different root, we can change the interval on which we instruct Sage to look for a root. For example, if we want to find the root near 1.2, we could try something like

```
find_root(x^5-3*x^4+x^3+2==0, 1, 2)
```

which gives us the value

```
1.199258801379252
```

Try to find the approximate value of the root of $x^5 - 3x^4 + x^3 + 2$ near $x = 2.5$.

(value is 2.563623765649018)

Note that we said that these values are approximate. Let's verify what we mean by that. Recall that a root of a function $f(x)$ is a value of x that makes $f(x)$ equal to exactly 0, i.e. $f(x) = 0$. Use Sage to define the function $f(x) = x^5 - 3x^4 + x^3 + 2$, then plug in the values of the approximate roots from above, e.g. find $f(-0.7931397744702121)$. If -0.7931397744702121 is truly a solution to $x^5-3x^4+x^3+2 = 0$, then $f(-0.7931397744702121)$ should equal exactly 0.

However, the value that we get from Sage is

```
1.35419453428653e-13
```

The e here is used for engineering notation, where 1.35419453428653e-13 means

$$1.35419453428653 \times 10^{-13}$$

In other words, the part before the "e" is a decimal number, but then the part after the "e" is the exponent that we should raise 10 to then multiply the decimal. Another way to think of this is that the "e-13" means we should start with the 1.35419453428653 then move the decimal to the left 13 times, making our number smaller. In contrast, if we had seen

```
1.35419453428653e5
```

that would mean move the decimal to the right 5 places, resulting in the number 135419.453428653.

The number $1.35419453428653 \times 10^{-13}$ is a really small number, but it's not exactly equal to 0. This demonstrates that the decimal "solution" we obtained using **find_root** is only approximate and not an exact solution.

Another thing to be cautious about when using **find_root** is that it will quit as soon as it finds one approximate root. In other words, even if there is another root in the interval that you specify, **find_root** will only tell you the value of one of them. If we try

```
find_root(x^5-3*x^4+x^3+2==0, 1, 3)
```

in hopes of finding both roots that are between 1 and 3, it will not give us both. Try this out and see what you get!

(only the solution 2.563623765649033 is found)

3.4 Iterating functions and chaos

We now consider what happens when applying the function $g(x) = -rx^2+rx$ to a number repeatedly.

We will start when $r = 0.5$, so that $g(x) = -0.5x^2 + 0.5x$. We will start with the value $x = 2$. What is the value of $g(2)$? $g(g(2))$? $g(g(g(2))$? If we continue to apply g more and more times to the result, what eventually happens to the output? Use Sage to make your calculations. You may want to apply **n()** to your output to get a decimal value to more easily see the pattern.

Now change x to any number of your choice and repeat the experiment of applying g repeatedly. What is the eventual outcome?

(all initial values converge to 0)

Now consider when $r = 1.2$, i.e. $g(x) = -1.2x^2 + 1.2x$. Start with an initial value of your choice and repeat the experiment of applying g repeatedly. What is the eventual outcome?

(all initial values converge to $\frac{r-1}{r} = 0.1\bar{6}$)

To track the results of iteratively applying the function $g(x)$ to an initial value of $x = a$, we can create a list of values.

```
var('x')
g(x)=-0.5*x^2+0.5*x

a=0.5

points = [a, g(a), g(g(a)), g(g(g(a))), g(g(g(g(a)))), g(g(g(g(g(a)))))]
```

We can then plot this using the **list_plots** function.

```
list_plot(points)
```

If we want to change the value of a, we can easily do so, then re-run the commands that come after it. In addition, if we want to see what happens when we apply $g(x)$ more times, we can also add those extra iterates to the list to get a better visualization of what happens to the output as we apply $g(x)$ more and more times.

Play around with different values of r between 2 and 4 and any choice of the initial value x. What patterns do you notice?

Now let's fix a value of $r = 3.5$, so we are looking at the function $g(x) = -3.5x^2+3.5x$. Pick two different values of x that are close to each other (within 0.1 of each other) and

plot repeated iterates of $g(x)$ on separate graphs. What do you notice? What happens as the two values of x get closer and closer together?

We note that the values seem to be clustered around 4 different values, and that the iterates seem to cycle through the values systematically. This is called a *periodic orbit*, and in particular, this is a periodic orbit of length 4 because the values (approximately) repeat every 4 times that we apply g. To find the periodic points, we can solve for when $x = g(g(g(g(x))))$. Find the points in this periodic orbit by solving $x = g(g(g(g(x))))$ using Sage.

(Solutions are $x = 0, \frac{3}{7}, \frac{5}{7}, \frac{6}{7}$)

4 Spreadsheets

4.1 Libré Office Calc

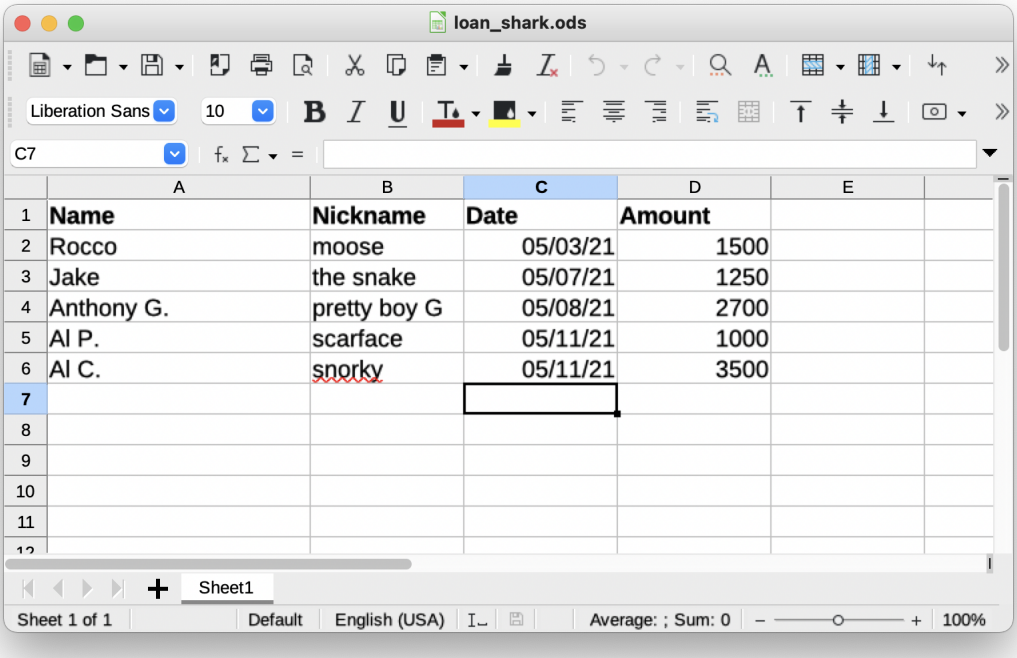
4.1.1 Introduction

tech

Spreadsheets were one of the original "killer apps" when personal computers were first introduced. Spreadsheets allow you to deal with tables of numbers and other data. The standard convention is that the rows of a spreadsheet are indexed by numbers and the columns are indexed with letters. If you need to go past 26 columns, use AA, AB, AC, et cetera for the next several columns. It would be pretty unusual to need to have more than 702 columns but if you needed to, guess what comes after ZZ.

Suppose a businessman was in the completely legitimate business of making loans to people who are regarded as poor credit risks by conventional banks. Of course, he'd want to keep track of the loan amounts and the recipients. A spreadsheet is the perfect tool!

Here's a screen shot of how the data might be recorded:



	A	B	C	D	E
1	**Name**	**Nickname**	**Date**	**Amount**	
2	Rocco	moose	05/03/21	1500	
3	Jake	the snake	05/07/21	1250	
4	Anthony G.	pretty boy G	05/08/21	2700	
5	Al P.	scarface	05/11/21	1000	
6	Al C.	snorky	05/11/21	3500	
7					
8					
9					
10					
11					

We're using the free/open-source Libré Office spreadsheet in the above. There are other choices (Excel on Windows and Numbers on Apple computers) but all of them work in a similar way and Libré Office is free. Spreadsheet files created by Libré Office have a .ods suffix.

The stuff that you enter into a cell in a spreadsheet fall into two main categories: a cell can contain data, or a cell can contain a calculated value. To make a "calculation"-type cell you have to put an equals sign up front. When you're doing a calculation you can use the values that are in other cells by referring to them by column (letter) followed by row (number). For example the $3500 that Snorky borrowed is in cell D6.

tasks

Open the loan shark spreadsheet (the file is available on the book website) and make some additions – columns for monthly interest rate, due date and the amount due.

Create a spreadsheet for keeping track of student grades in a math class. Use your favorite actors, sports stars, musicians (or whatever) as students, and just choose random numbers to put in as their grades. Create subtotals for homework, quizzes and exams, also a grand total with homework weighted 20%, quizzes weighted 30% and exams weighted 50%.

4.1.2 Absolute and relative cell references

tech

If you're going to be using the same calculation in a bunch of different places, you can just copy and paste the contents of one cell into another. Get used to using the keyboard shortcuts Ctrl-C and Ctrl-V for copy and paste.

When you copy and paste a formula, the spreadsheet intelligently changes the cell references in the formula. The pasted formula refers to cells that are in the same positions relative to the spot we're pasting into.

For example, if you type `=B3 + C2` into cell `C3` you're telling the system to add the number just above and the number just to the left. If you copy and paste that formula into cell `K7` you'll find that the formula has become `=J7+K6` because those cells are in the same relative positions.

This intelligent pasting usually does the right thing, but occasionally we really just want the thing to stay put! If you want a cell reference to not change when you're cutting and pasting (this is known as an absolute reference) put dollars ($) in front of both the letter and the number. Very occasionally we want a sort of hybrid behavior – we can put the dollar on one but not the other. For instance, if in a formula we refer to a cell using $A3 (with the dollar on the A but not in front of the 3) when we copy and paste the A will stay an A, but the 3 will change appropriately. Some people call this making either the row or the column "sticky."

math

In today's activity we'll be looking at two mathematical concepts: binomial coefficients and difference tables.

Binomial coefficients are sometimes called choice counters. For example, given a set of 5 options how many ways can we select 3 of them? This would be the binomial coefficient $\binom{5}{3}$ which is equal to 10. To pronounce that symbol in English use the word "choose," so the symbol above is read as "five choose three." That notation for binomial coefficients can be a little confusing since many people assume the fraction bar just got left off! So be careful, $\binom{5}{3} = 10$, but $\left(\frac{5}{3}\right) = 1.666\ldots$ so (obviously) these are different – don't imagine fraction bars where they don't actually appear!

There is another notation for the same quantities using a capital letter C. To indicate "five choose three" in this notation write ${}_5C_3$.

So why are these choice counters called "binomial coefficients"? It turns out these numbers also appear when taking powers of a binomial – a polynomial with just two terms. Try computing $(x+1)^0$, $(x+1)^1$, $(x+1)^2$ and $(x+1)^3$. Really, only the last of

those is at all difficult! Anything to the 0 power is just 1, anything to the 1 power is itself, and the 2nd power just requires the FOIL rule!

Blaise Pascal – a French mathematician who was one of the founders of the field of probability – was probably the first to notice the pattern when you write these things next to one another.

$$1$$
$$x+1$$
$$x^2+2x+1$$
$$x^3+3x^2+3x+1$$

The pattern becomes easier to deduce if you remove all the powers of x (and the plus signs) and just concentrate on the coefficients.

$$1$$
$$1 \quad 1$$
$$1 \quad 2 \quad 1$$
$$1 \quad 3 \quad 3 \quad 1$$

Numbers on the outside of each row are always 1. Numbers in the middle of a row are just the sum of the two things above them.

The arrangement of binomial coefficients into this triangular array is called Pascal's triangle. Here's the first 5 rows:

$$1$$
$$1 \quad 1$$
$$1 \quad 2 \quad 1$$
$$1 \quad 3 \quad 3 \quad 1$$
$$1 \quad 4 \quad 6 \quad 4 \quad 1$$
$$1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1$$

Difference tables come from a very common form of analyzing a sequence.

Suppose we asked, "What comes next?" in the following sequence.

$$4, \quad 7, \quad 10, \quad 13, \ldots$$

You probably notice rather quickly that successive terms in the sequence differ by the same number. A difference table is just a formalized way of making the same observation – except that if the differences don't seem to have an obvious pattern we might continue on taking the differences of the differences!

Here's an example. Suppose you're given the following sequence of numbers.

$$2 \quad 5 \quad 12 \quad 23 \quad 38 \quad \ldots$$

A difference table looks like so:

$$\begin{array}{ccccccccc} 2 & & 5 & & 12 & & 23 & & 38 \\ & 3 & & 7 & & 11 & & 15 & \end{array}$$

SInce the differences don't exhibit an obvious pattern we continue on, obtaining

2		5		12		23		38
	3		7		11		15	
		4		4		4		

It looks as though the bottom row – which is known as the second differences is always 4. Can you use that to predict what the next term in the sequence will be?

tasks

Write out the expanded form of $(x+1)^5$.

Create a table of the binomial coefficients $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Revisit the grade spreadsheet from the previous section and insert a new row at the top. Put the "weights" for homework, quizzes and exams into cells in this row. (As the teacher you might want to experiment with different weighting schemes.) Are the final grades changed by much if we change the weights to 10, 20 and 70 percent respectively?

Use a spreadsheet to do a finite differences analysis of the following sequence:

$$1 \quad 3 \quad 7 \quad 13 \quad 21 \ldots$$

Find the "back diagonals" for the sequences of squares, cubes, 4th powers etc. (We care about back diagonals because you can use them to generate the entire sequence! (under the assumption that the bottom-most number is a constant) (sorry about all the parentheses)).

Maybe: Given that the back diagonal for x^n contains $s(n,k) * k!$ in the kth row, and the recursion for the Stirling numbers. Use a spreadsheet to create a table of the Stirling numbers.

4.1.3 Builtin functions

Tasks:

Returning to our gradebook example...

Use the MIN function to "drop the lowest quiz."

Use functions to assign letter grades with + and − modifiers.

5 Geometry

5.1 GeoGebra

Bibliography

[1] Donald E. Knuth. The TEXbook. Addison-Wesley, 1984.

[2] Leslie Lamport. LATEX: A Document Preparation System. Addison-Wesley, 1986.

[3] William Strunk, Jr. and E. B. White. The Elements of Style. Macmillan, third edition, 1979.