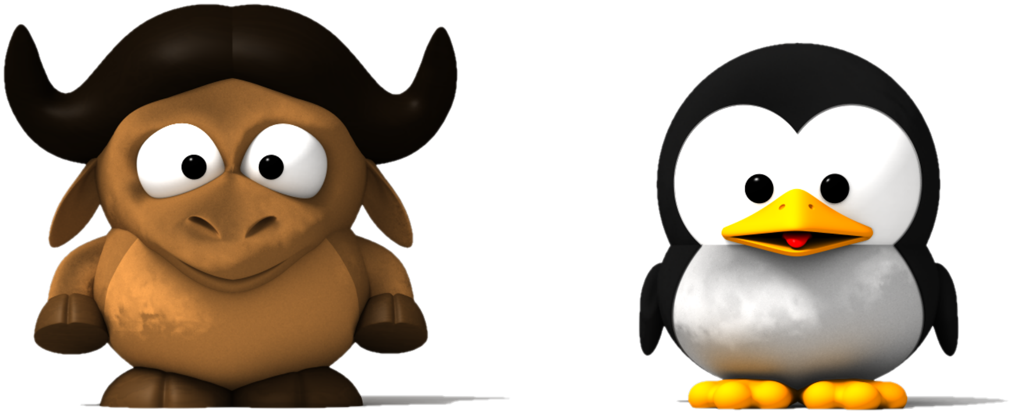


Serious Programming - small courses

Release 0.1.0

Mark Galassi, Leina Gries, Sophia Mulholland

Jan 11, 2024

CONTENTS:

Date
Nov 06, 2023

Author
Mark Galassi <mark@galassi.org>

Author
Leina Gries <leinagries@gmail.com>

Author
Sophia Mulholland <smulholland505@gmail.com>

Author
Almond Heil <almondheil@gmail.com>

CHAPTER

ONE

MOTIVATION AND PLAN

Section author: Mark Galassi <mark@galassi.org>

I cannot imagine a career more wonderful than that of a scientist.

The day-to-day work in science today involves using computers at all times. Scientists who master their computers and can program them with agility will enjoy the job the most, and are often in great demand: they can carry out unique new research. Young aspiring scientists who are not told this are being misled.

With an excellent group of young students, I have developed a series of lessons on scientific computing, aimed at kids who have already taken my "Serious Programming For Kids" course [Gal15]. I have two goals with these lessons:

a. introduce the tools and tricks for scientific computing, and

b. take a tour of diverse scientific problems that demonstrate "realy interesting" things you can do with some programming knowledge.

These mini-courses teach scientific computing using Python on the GNU/Linux operating system. There are other possible choices of programming language and operating system, and some of them are adequate, but there are specific reasons for which I chose Python and GNU/Linux. Some are given in the "Serious Programming for Kids" teacher's manual, but here are some other reasons which are specific to scientific work:

- Scientific software often matures into sophisticated programs which need to be executed on production computers and in a reproducible manner. For this the use of a free/open-source operating system and language interpreter are crucial.
- Much scientific infrastructure is available as an integral part of the GNU/Linux distributions. For example, on a current Debian GNU/Linux or Ubuntu or Fedora distribution you will find the GNU Scientific Library, astropy, scipy, a remarkable number of R science packages. These packages are "just there" as part of the operating system. This comes in part from the fact that the GNU/Linux operating system is developed by hackers for hackers: programming is a seamless part of such systems.
- Python spread rapidly soon after its initial development. Thanks to some key early developers coming from physics, astronomy and biology research groups, it was rapidly adopted by the scientific community. The result is that a vast collection of high quality scientific libraries are available in Python.
- Many research projects have very long lives, and the software is used for years after it is first written. My opinion, and that of many who observe the world of scientific computing, is that programs written in Python on a GNU/Linux system will still run many years from now[1]
- Reproducibility again: using proprietary software in scientific research makes it impossible to reproduce or verify a result: there is undisclosed code being executed!

[1] Programs written in the C programming language on a GNU/Linux system will be even more stable, thanks to the maturity and stability of the C standard. C is also a delightful and powerful language, but it is not in the scope of what I teach to younger kids. Only a few examples, those that need the higher speed of C, will be in C.

- Reproducibility and verifiability also dictate that scientific software should be able to run in *batch mode*, rather than through a graphical user interface (GUI). A GUI is not necessarily a bad thing, but after initial exploration of data with a GUI, the scientist needs to then generate a batch program to reproduce her results.

1.1 Notes for teachers

This is a teacher's manual for the mini courses. In the 10-hour "Serious Programming for Yough" workshop which introduces Python from scratch, I teach at a blackboard (or whiteboard nowadays).

This course is quite different: it is for students who have already taken the 10-hour workshop, and already have a laptop ready and running a GNU/Linux distribution.

The format is 1.5 hours, and I lecture with a projector or large TV screen, working on examples in emacs or in the command line.

While I lecture I have the students load the HTML version of this book, usually from a web site to which I sync this book – at this time I use http://markgalassi.bitbucket.io/ – this allows them to paste in code samples if they are too long to type.

I usually project a couple of terminals (one for python snippets, one for shell commands), a browser window with the relevant chapter of this book, and the emacs editor. This allows the students to see how I work on the examples.

The lecturing style should be one of quickly getting a juicy example up on their screens: something that gives visible results for the students. Then step back a bit to make sure they understood how we got to it, and then quickly on to the next example.

Once they have worked some juicy examples, it's time to lean back and have a broader discussion of the *meaning* of certain things, and to discuss the insight we got from an example. You can lace this with your favorite lecture on historical and philosophical aspects of what's in this chapter, but you should then quickly pivot back to more work. This "get back to work and roll up your sleeves" is a crucial part of what we do.

Understanding this material is hard work for the students: I have developed this course to include serious material they might otherwise not learn until college, so I often ask the students to "suspend their not understanding"[2] and just latch on to *one or two things they can remember*. For example I introduce Fourier Analysis in Section **??**, and when I give that lecture I frequently repeat "remember: it is OK to not understand most of this, but repeat after me the one thing I want you to understand: *all these signals look like wild jumbles, but they are made up of simple waves which let us understand part of their musical nature*."

In broad strokes you can think of two main categories of scientific computing effort: analyzing data from experiments, and simulating your own physical situation with a computer program that generates fake (but, we hope, realistic) experimental data. We will look at both of these types, and introduce the words: *experiments* and *simulation* as we go through the examples.

The way in which kids approach computers today (clicking and touching) allows them to not understand some concepts which are very important for scientific programming (and in fact any kind of programming). Because of this we must first get comformtable with the following concepts:

- What is a data file.
- How to plot a data file.
- How to write a program which takes a data file, does some processing of the data, and writes out another file with the processed data.

Once we have these skills we can:

- Tell the story of that plot.

[2] A pun on Coleridge's "suspension of disbelief" – with topics of great complexity it is important for students to be flexible about temporarily accepting a building block that they don't undersand so that they can keep with the flow.

- Generate simulated data.
- Retrieve data from online sources.
- Record data from an experiment.
- Analyze data to go *beyond that initial story*.

1.2 Acknowledgements

Thanks to Laura Fortunato and David Palmer for discussing this curriculum with me in detail before I developed it. Thanks to Jonathan Haack who has assisted me in teaching these courses and has given me feedback.

Thanks to my excellent Santa Fe students Lucas Blakeslee, Althea Foster, Alex Odom, Neha Sunkara, Rosa Birkner-Glidden, Miles Teng-Levy, Rowan Nadon, Teagan Boyes-Wetzel, Oisin O'Connell, Abby Wilson, Juan de la Riva, who have taken the course regularly and helped me develop it.

Most of all thanks to students and co-authors Leina Gries, Sophia Mulholland, Joaquin Bas for close collaboration on the book and for writing parts of it.

1.3 Status of the book

Some chapters are largely complete and just need polishing and proofreading; some have just a title; some are partially written.

Until the status is a bit more uniform, I will be putting a "readiness" status note at the top of the chapter. If you do not see such a status note then the chapter is probably not complete!

There is also an appendix on proposed chapters: Section **??**.

1.4 Footnotes

CHAPTER

TWO

STARTING OUT: DATA FILES AND FIRST PLOTS

[status: mostly-complete-needs-polishing-and-proofreading]

2.1 Motivation, prerequisites, plan

Data plotting and data visualization is the key

2.2 Very first data plots with gnuplot

Our first goal is to become comfortable with data files and with plotting. We first get the students to renew their acquaintance with creating files with an editor and make a file with some hand-crafted data.

Use your favorite editor (possibly `emacs` for those who have taken my previous course, but `vi` or `gedit` should also work) to open a file called `simpledata.dat`

Enter two columns of simple data into this file. For example:

```
1970   16.7
1980   17.2
1990   17.5
2000   18.2
2010   19.8
2020   20.1
2030   22.7
2060   28
```

Then save it, and enter `gnuplot` to plot this data:

```
$ sudo apt-get install gnuplot-x11
$ gnuplot
gnuplot> plot 'simpledata.dat'
```

then have the students plot the data with slightly different options in gnuplot:

```
$ gnuplot
gnuplot> plot 'simpledata.dat' with lines
gnuplot> set xlabel 'this is the "x" axis'
gnuplot> set ylabel 'this is the "y" axis'
gnuplot> plot 'simpledata.dat' with linespoints
```

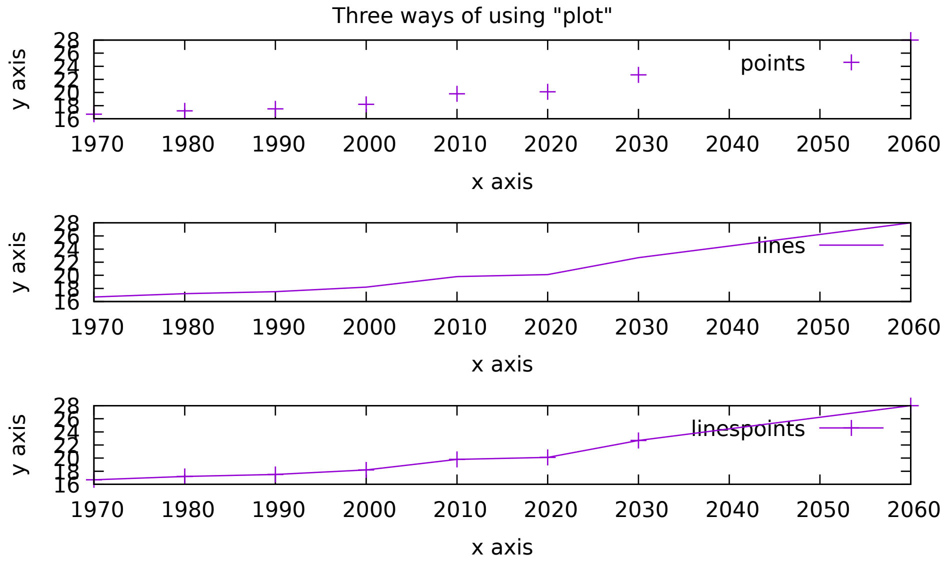


Figure 2.2.1: First example of data plot.

2.3 Plotting *functions* with gnuplot

More examples of using gnuplot. We don't assume knowledge of trigonometry from younger students, so we tell the story as "this is the sin function, which you will learn about some day; it plots these waves."

On the other hand the polynomial $y = x^3 - 3x$ should be within their reach: I might call on the class to tell me "what's -10^3 and -2^3, 2^3, and 10^3" - this establishes that the plot goes down on the left hand side, and up on the right hand side. The interesting play in the middle can be narrated by showing that $(1/2)^3$ is smaller than $3 * (1/2)$, so that the negative term dominates briefly (see Figure **??**)..

```
gnuplot> plot sin(x)
gnuplot> plot x*x*x - 3*x
gnuplot> plot x**3 - 3*x
## note that this last one did show a dip in the middle,
## but zooming in on the range from -3 to 3 shows the
## interesting features in the middle of the plot.  This
## plot has two flat points (one local maximum and one
## local minimum), rather than one saddle point.
gnuplot> plot [-3:3] x**3 - 3*x
```

If I am getting a good mathematical vibe from the classroom I will briefly step into calculus territory and ask students to predict the local max and min for $y = x^3 - 3x$. I will then quickly show them that

$$\frac{d(x^3 - 3x)}{dx} = 3x^2 - 3$$

and using the quadratic formula

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

we get $\pm 6/6 = \pm 1$, which is exactly where the local max and min are located (see Figure **??**).

You can now set the grid in the plot and see if this calculation matches the plot:

Listing 2.3.1: Instructions to plot a simple function and its derivative, in this case the function $f(x) = x^3 - 3x$ and its derivative $df(x)/dx = 3x^2 - 3$.

```
##CAPTION: Simple line plot.
set grid
## while we're at it also show the derivative:
plot [-3:3] x**3 - 3*x, 3*x**2 - 3
```

The two plots, superimposed in Figure **??**, show that where the derivative function ($3x^2 - 3$) is zero, the original function ($x^3 - 3x$) has its flat point. This can be presented, especially to older kids, in a rapid way that conveys "you don't have to understand this, but if you have heard about slopes then note that the second curve shows the slope of the first one…" For the younger kids we can emphasize that the figure shows two functions and looks intriguing.

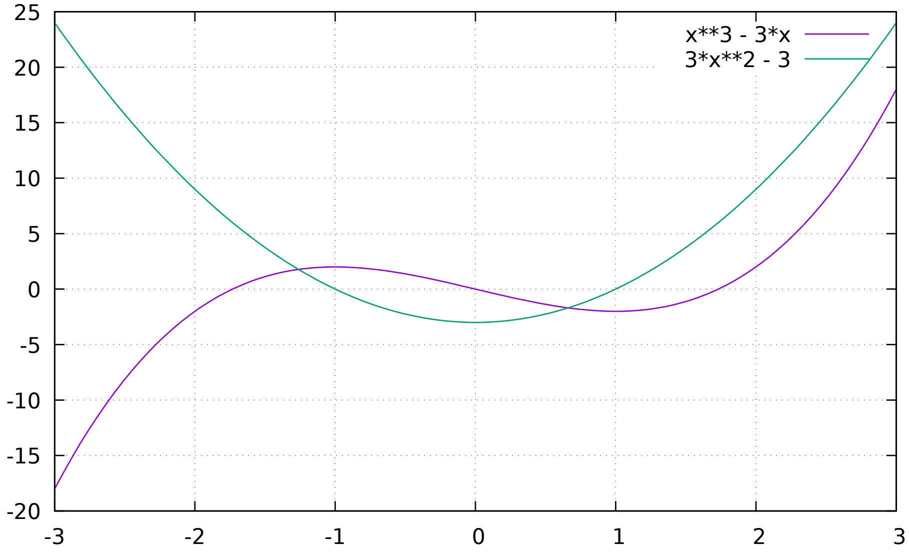


Figure 2.3.1: A simple plot of the function $y = x^3 - 3x$ and its derivative $y = 3x^2 - 3$.

Visualizing functions like this is cool and it can be useful during the exploratory phase of a research project, but it seldom comes up in the bulk of the work and in the final scientific write-up. We will now move on to tasks which come up quite frequently.

2.4 Reading and writing files, in brief

First let us make sure we know a couple of shell commands to look at a file. Here I usually will take a portion of the board and write a boxed inset "cheat sheet" with some useful shell commands.[4]

Since we already have a file called `simpledata.dat` which we created earlier, let us look at three shell commands that give us a quick glance at what's in that file: `head`, `tail` and `less`.

```
$ head simpledata.dat
$ tail simpledata.dat
```

(continues on next page)

[4] In the introductory course I have insets on the board with shell commands, `emacs` keybindings, and some Python commands. The `emacs` keybindings are especially important since the students have not necessarily done the full tutorial.

(continued from previous page)

```
$ less simpledata.dat
## (when using less be sure to quit with 'q')
```

These are simple ways to peek at a file, and will work with any text file. You should always remember these commands.

Next we will look at how to read a file in a Python program. This is a crucial pattern and we will use it a lot. Type in the program `reader.py` shown in Listing **??** and run it to see what happens.

Listing 2.4.1: reader.py – Reads a simple set of 2-column data

```
#! /usr/bin/env python3

"""show a simple paradigm for reading a file with two columns of
data"""

def main():
    fname = 'simpledata.dat'    # the file we wrote out by hand
    dataset = read_file(fname)
    print('I just read file %s with %d lines' % (fname, len(dataset)))
    print('I will now print the first 10 lines')
    N = min(10, len(dataset))
    for i in range(N):
        print(dataset[i])

def read_file(fname):
    dataset = []
    f = open(fname, 'r')
    for line in f.readlines():
        words = line.split()
        x = float(words[0])
        y = float(words[1])
        dataset.append((x, y))
    f.close()
    return dataset

main()
```

Remember that after writing and saving the program you do the following to make it executable and then run it:

```
$ chmod +x reader.py
$ ./reader.py
```

Finally let us see how to write files to disk. We will extend to do an easy manipulation of the file `simpledata.dat` and then write it back out to a new file `simpledata.dat.sums`. This new program will be called `simple-writer.py`, so we need to copy it first:

```
$ cp reader.py simple-writer.py
```

and edit `simple-writer.py` to look like Listing **??**.

Listing 2.4.2: simple-writer.py – Reads a simple set of 2-column data and sums the second column and then writes out line with (x y sumy).

```
#! /usr/bin/env python3

"""show a simple paradigm for writing a file after reading it and
adding some content to it"""

def main():
    fname = 'simpledata.dat'    # the file we wrote out by hand
    dataset = read_file(fname)
    print('I just read file %s with %d lines' % (fname, len(dataset)))
    print('I will now print the first 10 lines')
    for i in range(10):
        print(dataset[i])
    print('I will now modify the data')
    summed_data = append_sums(dataset)
    write_file(fname + '.sums', summed_data)
    print('I wrote the modified data to %s' % (fname + '.sums'))

def read_file(fname):
    dataset = []
    f = open(fname, 'r')
    for line in f.readlines():
        words = line.split()
        x = float(words[0])
        y = float(words[1])
        dataset.append((x, y))
    return dataset

def append_sums(dataset):
    sum_y = 0
    summed_data = []
    for pair in dataset:
        sum_y = sum_y + pair[1]
        triplet = (pair[0], pair[1], sum_y)
        summed_data.append(triplet)
    return summed_data

def write_file(fname, dataset):
    f = open(fname, 'w')
    for triplet in dataset:
        f.write('%g   %g   %g\n' % triplet)
    f.close()

main()
```

At this point we are comfortable with basic programming patterns for reading and writing data. This is very important: this kind of file manipulation is one of the big steps in becoming a confident programmer. This is a good time to make the kids familiar with the terminology "input/output" (I/O).

2.5 Generating our own data to plot

Now we look at an example of writing a python program to generate some data which we will then plot. Initially this just feels like silly data: it calculates the sin function for many opints and prints out the values. There is a reason to start with this very simple model: it will allow us (in the more advanced course on Fourier analysis) to give clear cut examples of deeper data anlysis. We will not be lazy: in the more advanced courses we will look at real signals instead of the toy ones.

Start with the `python3` command line and let us see the few lines of code that generate sin wave data:

Listing 2.5.1: simplewave.py – simple sin wave generator. You can run it with `python3 simplewave.py`

```python
#! /usr/bin/env python3

"""Generate a simpole sine wave"""

import math

for i in range(20):
    x = i/10.0
    signal = math.sin(x)
    print('%g    %g' % (x, signal))
```

```
0    0
0.1    0.0998334
0.2    0.198669
0.3    0.29552
0.4    0.389418
0.5    0.479426
0.6    0.564642
0.7    0.644218
0.8    0.717356
0.9    0.783327
1    0.841471
1.1    0.891207
1.2    0.932039
1.3    0.963558
1.4    0.98545
1.5    0.997495
1.6    0.999574
1.7    0.991665
1.8    0.973848
1.9    0.9463
```

Comments on this:

- This just prints values to the terminal, so we limited it to some 20 values of x. When we write it to a file for plotting we will do much more.
- The classic `for` loop generates a sequence of integers, which does not do well at all for plotting: we want to space our x values much more tightly to get a nice plot, so we divide the integer `i` by 10.0 to get finely spaced values of x.

Let us now put this into a file called `simplewave-write.py`

Listing 2.5.2: simplewave-write.py – simple sin wave generator, writes output to a file called `simplewave-write.dat`

```
#! /usr/bin/env python3

import math

def main():
    out_fname = 'simplewave-write.dat'
    f = open(out_fname, 'w')
    for i in range(200):
        x = i/10.0
        signal = math.sin(x)
        f.write('%g    %g\n' % (x, signal))
    f.close()
    print('finished writing file %s' % out_fname)

main()
```

We run the program and do a quick scan of its output with:

```
$ python3 simplewave-write.py
$ ls -l simplewave-write.dat
$ head simplewave-write.dat
$ tail simplewave-write.dat
```

and when we have seen the first and last few lines of the output file we realize we can plot it (after going back to our gnuplot window) with:

Listing 2.5.3: Instructions to plot the output of `simplewave-write.py`

```
##CAPTION: Plot the simple sin wave from a file.
set grid
plot 'simplewave-write.dat' using 1:2 with linespoints
```

Describing what this program does is rather straightforward, and it can be compared to the gnuplot instruction `plot sin(x)`.

At this point, at the blackboard, I will suggest to the students a couple of edits they "should have already tried on their own":

- See what happens if we don't divide $i/100.0$: try both `x = i` (for this use `range(7)`) and some intermediate value `x = i/4.0` (for this use `range(28)`). Note the loss of resolution.
- Use python's `sys.argv` to take the file name as a command line argument.

We then show a cleaner version of this program which adds some comments, makes clear what the sin period is, and uses some robust proramming paradigms such as using command line arguments.

Listing 2.5.4: simplewave-cleaner.py – More elaborate version of simplewave-write.py

```
#! /usr/bin/env python3

"""
Generate samples of a simple sin() wave and save them to a
```

(continues on next page)

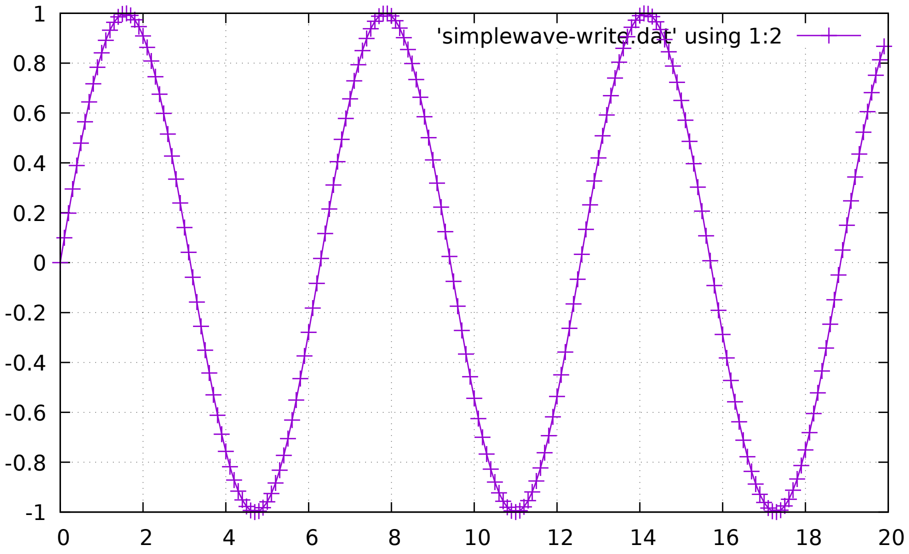


Figure 2.5.1: Plot of `simplewave-write.dat` which was output by our simple wave generator.

(continued from previous page)

```
file.  The file has two columns of data separated by white
space.  To run the program and plot its output try:
$ python3 generate-sin-cleaner.py 'sin_output.dat'
$ gnuplot
gnuplot> plot 'sin_output.dat' using 1:2 with lines
"""

import math
import sys

def main():
    out_fname = 'simplewave-write.dat'  # default output filename
    if len(sys.argv) == 2:
        out_fname = sys.argv[1]
    elif len(sys.argv) > 2:
        print('error: program takes at most one argument')
        sys.exit(1)

    f = open(out_fname, 'w')
    for i in range(700):
        x = i/100.0
        signal = math.sin(x)
        f.write('%g    %g\n' % (x, signal))
    f.close()
    print('finished writing file %s' % out_fname)

main()
```

Note that the comments also tell you how to run a program and visualize the output. This use of comments to explain behavior is an important detail, even for small programs.

In the more advanced course on Fourier analysis we will revisit this simple program and make it generate more complex waveforms.

2.6 The broad landscape of plotting software

Now that we have seen some examples, let us talk broadly about plotting software, since you will soon be bombarded with people telling you about their favorites.

By now we know well that in the free software world there is often a dizzying variety of tools to do any task, with fans advocating each approach. Gnuplot is full-featured, stable, actively maintained and ubiquitous so I have chosen it, there are several other valid choices.

There are at least three main categories of plotting tools:

a. The "just a plotting program" kind,

b. The "plotting program with some data analysis that grew into a full programming language",

c. The "plotting library for a well-established programming language".

Gnuplot is clearly one of the first, R and Octave the second, Python with Matplotlib and Cern's Root are examples of the third.

Often it comes down to where a particular scientist did her early research work: a boss will tell you to "use this tool because it's what I use". I recommend forming a broad knowlege of scientific tools so that you can use *the most appropriate tool* instead of *the tool that makes your boss comfortable*. You will often find that astrophysicists often use Python with matplotlib, particle physicists use Cern's Root, biologists use R or Python, social scientists who do much statistical work use R. You shoud always know the tool your community uses (if it's a free software tool), as well as some others which might be more appropriate.

There are many proprietary plotting packages. I advocate against the use of proprietary software, and it is certainly unacceptable to do science with proprietary tools, but I will mention a couple of packages so that when you come across users you will be able to categorize and compare them and offer an effective free software approach to the same problem.

CricketGraph, often used in the 1990s, was a light-weight plotting program, later supplanted by KaleidaGraph. I would recommend gnuplot as a good way of doing what those programs did.

Matlab and IDL are simple plotting and data analysis programs that grew out of control and added an ad-hoc programming language to the package. The languages were never meant for large software applications, but are often used to write very large programs. It is interesting to note that these programs always start with the stated intention of not requiring a scientist to know how to program, but they end up channeling scientists into using an ill-designed language for large programs.

Matlab and IDL programs can be written in a much cleaner way using Python for the programming parts, and the matplotlib plotting library for graphics.

There is a final outlier in the proprietary data analysis world, which is the "using a spreadsheet to do data analysis" approach, often with the proprietary Excel spreadsheet. There is no saving grace to this approach: apart from technical concerns with the validity of the numerical subroutines, there is also the complete lack of reproducibility of a person moving a mouse around over cells. One of the most embarrassing cases of incorrect analysis was in a much-cited Economics paper about debt rations in European countries [RR10]. The analysis was done with an Excel spreadsheet, and some readers concluded that the authors selected the wrong range of data with a mouse movement. There is no reproducibility when mouse movements are part of the data analysis. The economics article was disgraced because of the faulty analysis as well as other problems with their methodology.

2.7 Data formats

In Section **??** and Section **??** we saw the simplest examples of data files: columns of numbers separated by white space. These are the simplest to work with, and if your files are smaller than about 10 megabytes you should always treat yourself to that simplicity. This format is often called "whitespace separated ascii" or names similar to that.

Often you will find that the columns of data are separated by commas. This format is called "comma separated values" (csv) and the files often end with the `.csv` extension. The format has been around almost half a century. It has some advantages over the whitespace separated columns and is used by almost all spreadsheet programs as an import/export format. In gnuplot you handle this with the instruction `set datafile separator comma` as we see in Section **??**.

Sometimes files are in a variety of *binary formats*, which you cannot read directly. We will not deal with these at this time, since we are not yet working with very big files, but later on we will show how to convert mp3 files to an ascii format which is easily read by our programs and by gnuplot.

2.8 Simple surface plots

So far we have looked at *line plots*. Let us now look at another type of plot: the *surface plot*. A surface plot comes from a function of two variables: $z = f(x, y)$. In these plots the value of the function is plotted as the *height* over the x, y plane. Here is an example:

Listing 2.8.1: Instructions to plot a simple surface, in this case the function $f(x, y) = e^{\frac{-x^2-1.7y^2}{10}}$

```
##CAPTION: Simple surface plot.
set grid
set pm3d
set xlabel 'x'
set ylabel 'y'
set samples 50
set isosamples 50
splot exp((-x*x-1.7*y*y)/10.0) with pm3d
```

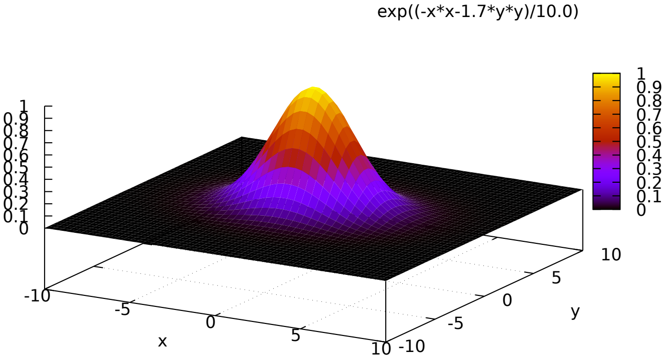


Figure 2.8.1: The surface plot of $f(x, y) = e^{\frac{-x^2-1.7y^2}{10}}$

The plot in Figure **??** is visually gratifying, and even more so because when you generate it interactively with gnuplot you can grab it with the left mouse button and rotate it to get more insight into what the surface looks like from different angles.

Another way of showing the same information is a *heat map*. Figure **??** shows the same function, but this time the value of the function is represented with *color* instead of height.

Listing 2.8.2: Instructions to plot a heat map, in this case the function $f(x,y) = e^{\frac{-x^2-1.7y^2}{10}}$

```
##CAPTION: Heat map plot.
set size ratio -1
set view map
set samples 50
set isosamples 50
set xlabel 'x'
set ylabel 'y'
splot exp((-x*x-1.7*y*y)/50.0) with pm3d
```

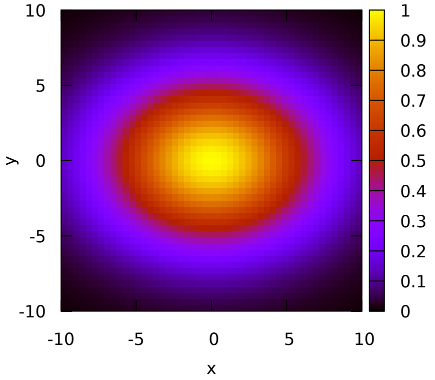


Figure 2.8.2: The heat map for of $f(x,y) = e^{\frac{-x^2-1.7y^2}{10}}$

2.9 Topics we have covered

- data files
- plots
- gnuplot
- reading features from simple plots
- simple surface plots

CHAPTER

THREE

INTERMEDIATE PLOTTING

[status: mostly-written]

Motivation

Plotting is maybe the main tool scientists use to develop *their own* insight into what they study, and is also the main tool they use to communicate their results and insights to others.

We continue to study plotting, introducing *histograms* and then learning how to plot directly from Python with matplotlib.

Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**.

Plan

We will:

1. Work through an example which motivates the introduction of histograms and of plotting from a program.
2. Introduce histograms.
3. Learn how to make plots directly in a python program with matplotlib.

3.1 A worked example

Histograms, bins, distributions… What are these? What kind of insight do they give?

Let's start by looking at a data set on human height and seeing what we can do with it. We will use the Howell census data for the !Kung people of the Kalahari desert. I will guide you through trying to pull information out of this file, paying attention to where we introduce new ideas and techniques.

Download the file with

```
$ wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/Howell1.csv
$ wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/Howell2.csv
```

Looking at these files we see that (a) they have the `.csv` extension, meaning that they are “comma separated values”, but (b) looking at the contents with `head Howell1.csv` or `less Howell1.csv` shows that the data fields are separated by semicolons instead of commas. We need to know this to give the right plotting instructions.

Weight vs. height

Taking the “what do you do with witches” approach, let us immediately plot the data in this file:

```
$ gnuplot
gnuplot> set datafile separator ";"
gnuplot> plot 'Howell1.csv' using 1:2
```

What have we just done? we plotted the first two columns, and the file header tells us that those columns are height and weight, so let us make the plot more clear like this:

```
gnuplot> set datafile separator ";"
gnuplot> set xlabel "height (cm)"
gnuplot> set ylabel "weight (kg)"
gnuplot> plot 'Howell1.csv' using 1:2
```

What do we call this kind of plot? It’s a “scatter plot”: the height is not in any particular order, and there can be variability in weight for a given height, so there is no point in a line plot. That’s why we plot points for scatter plots.

Insight: taller people weigh more, but there is some variability (people can be overweight or underweight).

Terminology: scatter plot.

Terminology: we have plotted “weight versus height” or “weight as a function of height”.

Height vs. age

What more can we do? The first line tells us we also have age in column 3, so let’s look at height as a function of age:

```
gnuplot> set datafile separator ";"
gnuplot> set xlabel "age (yr)"
gnuplot> set ylabel "height (cm)"
gnuplot> plot 'Howell1.csv' using 3:1
```

Let’s add a grid to this one:

```
gnuplot> set grid
gnuplot> plot 'Howell1.csv' using 3:1
```

Type of plot: scatter plot of height vs. age.

Insight: you grow until age 20, then you stop growing. You might lose some height when you are really old.

Weight vs. age

We can guess that weight vs. age will look a bit like height vs. age, but check it out:

```
gnuplot> set datafile separator ";"
gnuplot> set xlabel "age (yr)"
gnuplot> set ylabel "weight (kg)"
gnuplot> set grid
gnuplot> plot 'Howell1.csv' using 3:2
```

Insight: as the !Kung people get older they weigh more until their 20s, then it stabilizes.

Distinguish female and male

The header in `Howell1.csv` tells us that column 4 is 1 for male and 0 for female. What do we expect to see if we somehow distinguish the male and female parts of the plot?

Let's first make some guesses in our mind about what we should see. Would weight vs. height be significantly different for females and males? How about height vs. age and weight vs. age?

Now let's plot it:

```
gnuplot> set datafile separator ";"
gnuplot> set ylabel "height (cm)"
gnuplot> set xlabel "sex (0=female, 1=male)"
gnuplot> set grid
gnuplot> plot 'Howell1.csv' using 4:1
```

If you peer at this plot closely you will see that the points are concentrated in a higher band for males than for females, but there are problems: (a) it's deeply inelegant, (b) this visual insight could easily disappear with much more data, and (c) you cannot deduce any specific quantities from this plot.

We will look at a couple of ways in which this can be plotted better using the tools we have, but in the end we will realize that we are forcing our tools a bit, which is a sign that we need new tools.

First we try to split the file into two separate pieces. For this file we can do it with the ever-amazing tool `grep`. Note that the lines for females have the snippet `;0` at line's end and those for males have `;1` at the end of a line, and now type:

```
$ grep ';0$' Howell1.csv > females.csv
$ grep ';1$' Howell1.csv > males.csv
$ gnuplot
gnuplot> set datafile separator ";"
gnuplot> set xlabel "age (yr)"
gnuplot> set ylabel "height (cm)"
gnuplot> set grid
gnuplot> plot 'females.csv' using 3:1
gnuplot> replot 'males.csv' using 3:1
```

The `$` in the grep command matches the end of a line.

Insight: by age 20 the !Kung people have a height difference between men and women.

This took a bit of work. Another approach is to use a feature of gnuplot which colors the points according to the value of another column. This saves you from creating the two separat files. You can do it like this:

```
gnuplot> set datafile separator ";"
gnuplot> set xlabel "age (yr)"
gnuplot> set ylabel "height (cm)"
gnuplot> set grid
gnuplot> plot 'Howell1.csv' using 3:1:4 with points linecolor variable
```

This will show different colors for male and female.

Needing more tools

Both the approaches we saw to separate out female and male data are clumsy: the first one makes you create extra files, while the second one feels contrived and doesn't give you very good control over the plot.

There is also another problem with scatter plots. They give good rapid insight, like "between the heights of of 140cm and 175cm there is agreat variability in weight. But they do not allow you quantify it. For example: the middle of the jumble of points does not allow you to say how many points are in there and to distinguish different parts. This is especially true of larger data sets.

Examples of questions you could not answer too well with the plots we have:

- Which is the most common height among the !Kung people?
- Are most adults close to that average height or does it vary a lot?
- Can we see how those quantities vary for just grown-ups? Or just children of certain ages?

This leads us to introduce the new conceptual tool of the *histogram* and to discuss how to *plot from within a python program*.

3.2 Histograms

Looking at the question "hich is the most common height among the !Kung people?", let us try to answer it this way: for each height from 135cm to 180cm, how many people are that tall?

Of course we can't say "for each height": if you measure precisely enough there will be just one person for each height!

So we break up that range into *bins*, for example 135 to 137, 137 to 139, . . . , 173 to 175, 175 to 179, 179 to 181. There should be some 22 bins, each of which is 2cm wide.

Then for each one of these bins we add up how many people have that height.

Terminology: the *bins* are those 2cm spaces, the *bin edges* are the minimum and maximum values for the bin. The *bin width* is the 2cm between the high edge and the low edge.

To do this, enter the program in Listing **??**:

Listing 3.2.1: make-height-histogram.py – make a histogram from a file whose first column has heights.

```
#! /usr/bin/env python3

"""Takes file where the first column has heights and makes a histogram
out of it, putting them into 2cm bins between 130cm and 180cm.
"""

import sys
```

(continues on next page)

(continued from previous page)

```
import matplotlib.pyplot as plt
import numpy as np

def main():
    fname = sys.argv[1]
    heights = []
    ## define the "bin edges"
    bin_edges = [i for i in range(135, 182, 2)]
    n_bins = len(bin_edges) - 1
    ## initially all bins are empty
    histogram = [0] * n_bins
    ## now open the file and read in all the values in the first
    ## column; put them in bins as we read them
    with open(fname, 'r') as f:
        lines = f.readlines()
        for line in lines:
            if line[0] == '"':
                continue # skip the first line (it starts with a quote)
            ## break the line up into words, splitting on semicolons
            words = line.split(';')
            ## the first word is the height; convert it to float
            height = float(words[0])
            ## now put it in the right bin
            for bin, low_edge in enumerate(bin_edges[:-1]):
                if height >= low_edge and height < bin_edges[bin+1]:
                    ## found it!
                    histogram[bin] += 1

    ## now write out a file with the midpoints of the bins on the x
    ## axis and the number of heights in that bin on the y axis
    hist_out_fname = sys.argv[1] + '.hist'
    with open(hist_out_fname, 'w') as f:
        for bin in range(n_bins - 1):
            midpoint = (bin_edges[bin] + bin_edges[bin+1]) / 2.0
            f.write('%g    %d\n' % (midpoint, histogram[bin]))
            print('%g    %d' % (midpoint, histogram[bin]))
    print('wrote histogram to %s' % hist_out_fname)
    print('you could plot in gnuplot with the command')
    print('plot %s using 1:2 with boxes' % hist_out_fname)

if __name__ == '__main__':
    main()
```

Run the program with:

```
$ ./make-height-histogram.py Howell1.csv
```

Then plot the resulting histogram with the following gnuplot instructions:

Listing 3.2.2: Instructions to plot the height distribution plot for the !Kung.

```
##REQUIRED_FILE: Howell1.csv
##REQUIRED_FILE: Howell1.csv.hist
##REQUIRED_FILE: females.hist
##REQUIRED_FILE: males.hist
##PRE_EXEC: wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/
↪Howell1.csv
##PRE_EXEC: ./make-height-histogram.py Howell1.csv
##PRE_EXEC: grep ';0$' Howell1.csv > females.csv
##PRE_EXEC: grep ';1$' Howell1.csv > males.csv
set grid
set title 'height distribution of the !Kung people'
set xlabel 'height (cm)'
set ylabel 'number of people with that height
set style data histogram
set style fill solid 0.8 border -1
plot 'Howell1.csv.hist' using 1:2 with boxes
```

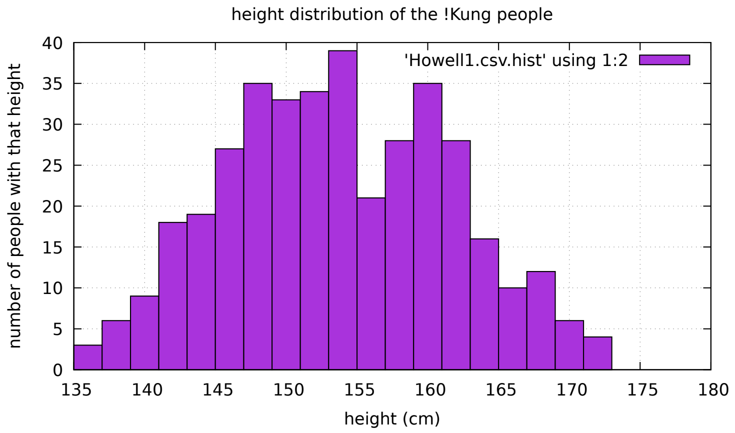


Figure 3.2.1: Histogram of how many people are in a given range of height. This is for the !Kung people of the Kalahari desert, and the data set includes both females and males.

This plot seems to show two humps, one around 152cm and one around 160cm. But wait: haven't we been told that height distribution should look like a bell shaped curve? This one does not.

So let us use the separate male and female data we obtained earlier and that is in the files `females.csv` and `males.csv`:

First we make height histograms for females and males:

```
$ ./make-height-histogram.py females.csv
$ ./make-height-histogram.py males.csv
```

Then plot with:

Listing 3.2.3: Instructions to plot the height distribution plot for the !Kung

```
##REQUIRED_FILE: Howell1.csv
##REQUIRED_FILE: Howell1.csv.hist
```

(continues on next page)

(continued from previous page)

```
##REQUIRED_FILE: females.hist
##REQUIRED_FILE: males.hist
##PRE_EXEC: wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/
↪Howell1.csv
##PRE_EXEC: ./make-height-histogram.py Howell1.csv
##PRE_EXEC: grep ';0$' Howell1.csv > females.csv
##PRE_EXEC: grep ';1$' Howell1.csv > males.csv
set grid
set title 'height distribution of the !Kung people'
set xlabel 'height (cm)'
set ylabel 'number of people with that height
set style data histogram
set style fill solid 0.8 border -1
plot 'Howell1.csv.hist' using 1:2 with boxes, \
    'females.csv.hist' using 1:2 with boxes, \
    'males.csv.hist' using 1:2 with boxes
```

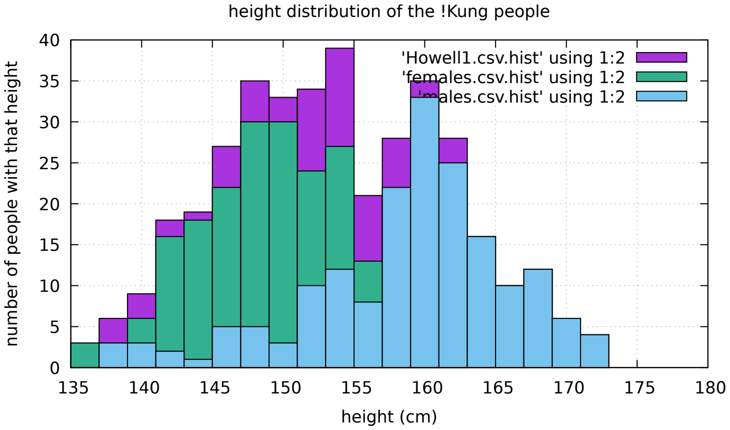


Figure 3.2.2: Histogram of how many people are in a given range of height. This is for the !Kung people of the Kalahari desert, and we also show the separate male and female data. The gaussian (bell-shaped) distribution is now clear.

We finally have what we were hoping for: a clear gaussian (bell-shaped) distribution of data around an average height. We had to separate male and female heights to get that.

Insight: these histograms of frequency of occurence of certain heights give us insight into the nature of human height.

To conclude: we have written a program which takes data and makes histograms out of it. It is useful to know how to do this, but we will see that this approach gets cumbersome (we have to run several different programs), so we will learn how to programs that use libraries to make histograms and to make plots.

Exercise 3.1 We only looked at heights greater than 135cm because we were interested in fully grown men and women. Do you see the problem here? Some children might pass through height 135cm before they reach full height, so we might get some data in there that is not appropriate. Adjust `make-height-histogram.py` to exclude people under the age of 20 from the histograms.

Exercise 3.2 Examine the CSV files at https://vincentarelbundock.github.io/Rdatasets/datasets.html using the approaches we have been using in this chapter.

3.3 Matplotlib

Matplotlib is a library for making plots *within a python program*. Using this we can manipulate data in a program and draw it right away, rather than always having to write out intermediate data files.

As with gnuplot, matplotlib allows us to make interactive plots or to write them out to graphics files (png, pdf, svg, …)

My opinion is that there are times when you want to use matplotlib and times when you want to write out text files and invoke gnuplot on it. Developing a feeling for what's appropriate is a part of developing your personality as a scientist. A starting point is:

- Use gnuplot (or another command line plotting program) for a quick exploration into a dataset, or for a reproducible command pipeline.
- Use python+matplotlib when you are doing lots of data manipulation before generating the plot, or when you are generating plots at various stages of processing.

Before anything else we must install matplotlib for Python3:

```
$ sudo apt install python3-matplotlib
```

In class we follow the matplotlib tutorial offered by the developers of matplotlib:

https://matplotlib.org/users/pyplot_tutorial.html

(but we should also take the example with the cute multicolored bubbles from the newer tutorial, even though it does not work on matplotlib 2.0.0 which comes with ubuntu 16.04. That example is at: https://matplotlib.org/tutorials/introductory/pyplot.html)

Then we can dip in to some of the further material at:

https://matplotlib.org/tutorials/index.html

in particular we can take a tour of the "Sample Plots in Matplotlib".

After doing this we can try some exercises:

Exercise 3.1 Redo all the work from earlier sections in this chapter using matplotlib instead of gnuplot.

Exercise 3.2 Make all the programs in the previous exercise take an optional command line option which is an output file for the plot. If there is no command line option then make the plot interactive.

3.4 A histogram snippet to conclude

Finally: I will reproduce here a useful snippet from the tour: how to make and plot histograms using numpy and matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt

## [...] collect quantity to be histogrammed in an array x

n, bins, patches = plt.hist(x, 50, normed=1, facecolor='g', alpha=0.75)

plt.xlabel('base quantity')
plt.ylabel('Probability')
plt.title('Histogram of base quantity')
```

(continues on next page)

(continued from previous page)

```
plt.grid(True)
plt.show()
```

CHAPTER

FOUR

A TOUR OF FUNCTIONS

Section author: Leina Gries <leinagries@gmail.com>

[status: content-mostly-written]

4.1 Motivation, Prerequisites, Plan

A function is a relationship between two (or more) variables. For now we will call one of them x and the other $y = f(x)$ (*the value of the function at* x). For this relationship to be a true function, each value of x must give a unique value of y.

We use other terminology for functions. We can say:

- y is a function of x
- f is a mapping from x to y

There are other terms people use as well. As usual, don't worry about differing terminology: they mean more of less the same thing: you take a value x, put it into a machine, crank the machine, and out pops the value y.

The goal of this chapter is to gain a deeper understanding of mathematical functions and how to go about plotting them. Initally, we will experiment with basic linear functions, using them as a tool to understand plotting software and seeing how visualization can be useful. We will experiment with plotting multiple lines at a time, and look at how the geometry matches up with the algebra.

Then we will look at polynomials and explore how they solve some physical problems and how the solutions to these problems can be visualized with plotting software.

Afterwards, we will move into the realm of more advanced functions- touring exponential growth and decay, the curves made by sine and cosine waves, bell curves (Gaussian Distribution), factorials and the fibonacci sequence. Then, we will cover some functions that create interesting and visually pleasing graphs.

Finally, we will apply our findings to the real world and plot functions that describe real life events, such as climate change and nuclear fission.

4.2 Linear Functions

To start, let's stay linear. Graph the equation $y = 5x - 8$ by running gnuplot and typing the following expressions:

```
gnuplot> set grid
gnuplot> plot 5*x - 8
```

This should bring up a graph displaying the equation in a seperate window. Try plotting this with and without grid lines, using the command (code). Explore the window. What can you do to make the line displayed change?

Can you make the slope negative?

Can you graph several lines at once? This brings us to plotting two linear functions. Try plotting the original line with the addition of $y = 4x - 6$.

```
gnuplot> set grid
gnuplot> plot 5*x - 8
gnuplot> replot 4*x - 6
```

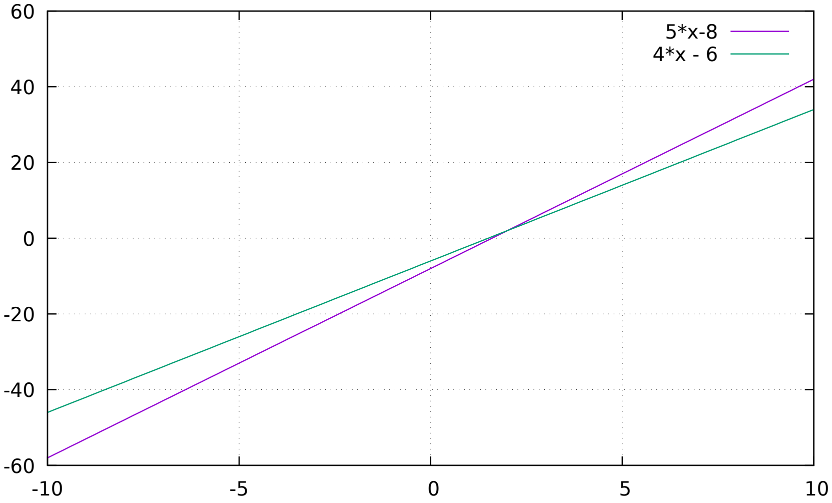


Figure 4.2.1: Two lines, one with slope 5 and the other with slope 4. Lines with different slopes will always intersect.

We see *visually* in Figure **??** that the lines intersect. How do we figure out at which point they touch? Visually we can see that x is near 2 and y is near 2, but can we get the exact values?

This is where the beauty of *analytic geometry* comes in: there is a relationship between the gometry (which we see visually) and the analysis that you can do using algebra. The key here is:

```
The intersection is the point at which the x and y values of both
lines are the same.
```

How do we express that algebraically? We can start by saying that if $y = 5x - 8$ and $y = 4x - 6$, then:

$$5x - 8 = 4x - 6$$

There are many ways of solving this, and depending on where you are in your math studies you might have seen one or more. I will do one of the simplest: if $5x - 8$ and $4x - 6$ are equal, then I can add and subtract *the same* quantities from both, and the result will still be equal. That gives

$$5x - 8 = 4x - 6 \tag{4.2.1}$$
$$5x - 8 - 4x = 4x - 6 - 4x \tag{4.2.2}$$
$$x - 8 = -6 \tag{4.2.3}$$
$$x - 8 + 8 = -6 + 8 \tag{4.2.4}$$
$$x = 2 \tag{4.2.5}$$
$$\tag{4.2.6}$$

Once we have x we can plug that value into either of the equations for y and we get $y = 2$, so the intersection point is $(2, 2)$

This technique can be used in many other circumstances to find where two *curves* intersect. The algebra will be more complicated than finding the intersection of lines, but the idea will be the same.

Can you think of a way to find out multiple points of intersection between three or more lines?

Another bit of terminology: these functions in which x appears only in the first power (there is no x^2 or x^3) always give straight lines when you plot them. Because of this we call them *linear* functions.

FIXME: discuss Direct proportionality.

4.3 Polynomials

Polynomials are the next step in complexity. Let us start with a binomial (second degree polynomial). Plot the function $y = 3x^2 + 6x - 4$

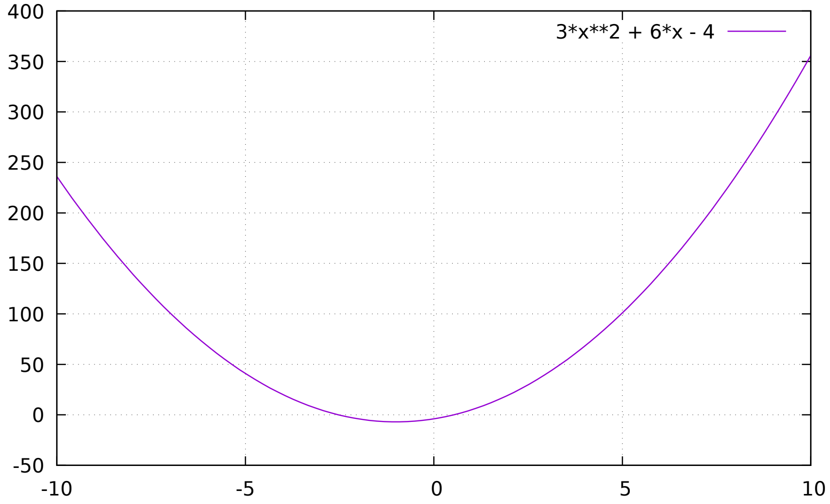


Figure 4.3.1: A simple second degree polynomial $y = 3x^2 + 6x - 4$

From the graph, what can you observe about this function? Where are the x intercepts? The y intercept? While these are clear in some cases, they are difficult to find in others.

The figure that comes from plotting a second degree polynomial is called a *parabola*. This one has two arms that grow up from its minimum.

We will spend a bit of time discussing this plot because it teaches us some interesting things to look out for.

- This polynomial has three *terms:* $3x^2$, $6x$ and -4. Each of these three terms has more or less importance according to the value of x. When x is 0, then the constant term 4 dominates. When x is between 0 and 1 or 0 and -1 then the term $6x$ grows to be more important. When x gets much bigger than 1 (or much smaller than -1) then the $3x^2$ term (second order term) dominates.
- Terminology: the number that multiplies x^2 (in our case 3) is called the *coefficient* of the second order term (or the *quadratic* term). The number that multiplies x (in our case 6) is called the coefficient of the *first order term* (or *linear* term). The term that doesn't have x is called the *constant term* or the *offset*.
- The arms of the parabola eventually point up. This is because the term with x^2 gets much bigger than the other ones, and when x is negative, x^2 is still positive.
- Zero crossings: it is interesting to note where the parabola crosses the x axis and the y axis. In class we discuss how to find this by zooming in on the picture (both with the mouse and with the range parameters in gnuplot). Terminology: the values of x at which a function crosses the x axis (i.e. where $y = 0$) are called the *roots* of the function.
- It is also interesting to note the value of x for which the parabola has its *minimum* or *maximum* values. Sometimes there is no *global* minimum or maximum, so in class I describe what a *local* minimum or maximum is. There is a technique for finding maximum and minimum values for a function. This involves calculating the *derivative* of the function. We will say a bit more about derivatives below.

4.3.1 Derivatives

Without introducing the entire subject of differential calculus (for that see Section **??**), let's look at the parabola we have been considering: $y = 3x^2 + 6x - 4$. Looking at the plot, study what the slope of that plot is for big negative x, for x between -3 and 0, and for big positive x.

We expect the slope to be big and negative for very big negative x, close to zero for that value of x between -3 and 0, and then to get positive and big.

When we learn calculus we will learn that you can figure out exactly what this slope is at every point in the curve. Without showing how it's done, I show the result here. We call the function that is the slope of $f(x)$ the *derivative* of f, and we write $\frac{df(x)}{dx}$. The derivative of this second degree polynomial is:

$$\frac{df(x)}{dx} = 2 \times 3x + 6 = 6x + 6$$

The properties of derivatives that we used here were: (a) the derivative of a sum of terms is the sum of the derivatives of each term, and (b) the derivative of x^n is $dx^n/dx = n \times x^{n-1}$.

The place at which the derivative is zero is $x = -1$, so we can say that the our parabola has its minimum at $x = -1$. If you zoom in to the plot enough you will see that this is true geometrically.

You should explore this polynomial further. You should change the sign or some of the values and see how different the plot looks.

4.4 Higher order polynomials

Having successfully plotted a binomial, let's move onto higher order polynomials. Plot $x^4 + x^3 - 7x^2 - x + 6$. The result will initially look somewhat boring.

Now, finding the root (or solutions, found at the x intercepts of the equation) are much more dificult to find manually.

By zooming in further, find the roots of this 4th order polynomial. How many are there? Does this relate to the degree of the polynomial?

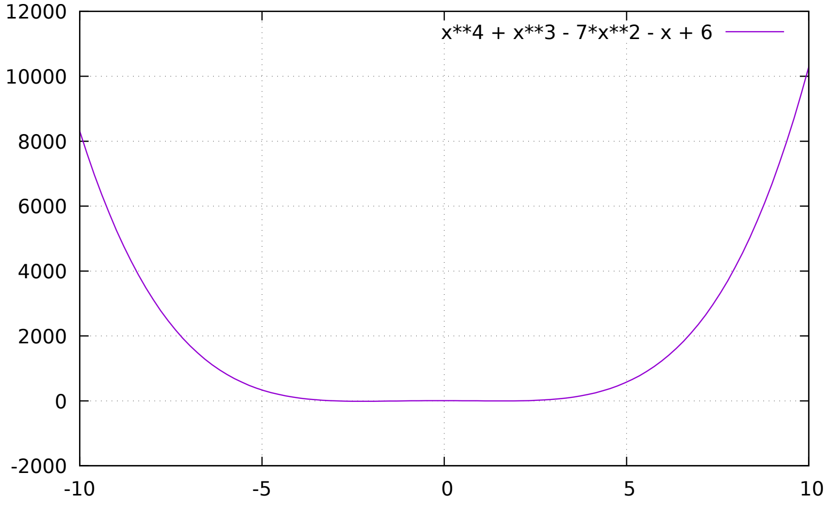


Figure 4.4.1: A fourth degree (quartic) polynomial $y = x^4 + x^3 - 7x^2 - x + 6$. This does not look too interesting at this time.

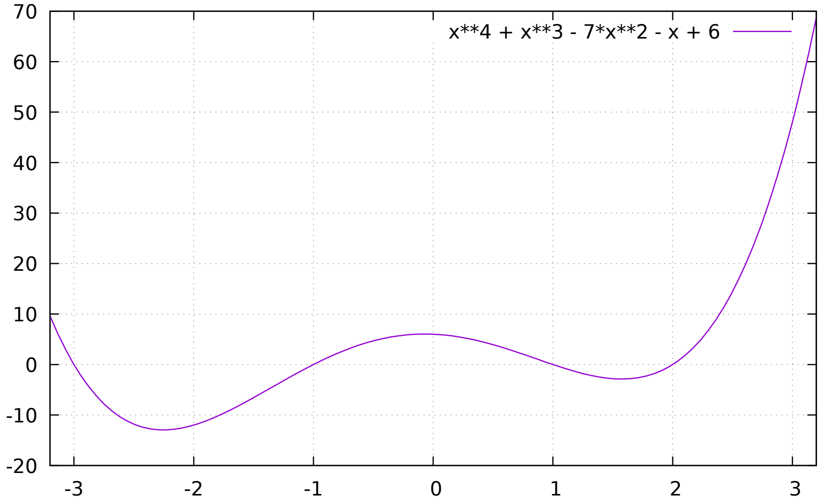


Figure 4.4.2: The same fourth degree (quartic) polynomial $y = x^4 + x^3 - 7x^2 - x + 6$. By zooming in to the x range [-3.2:3.2] we see interesting features.

And now let me tell you how I picked this polynomial: I wanted to find a polynomial that crosses the x axis at -3, -1, 1 and 2. How do you do that? First notice that the first order polynomial $(x-1)$ is zero when $x=1$. So we can craft a set of first 1storder polynomials (linear functions) which cross the x axis at -3, $(x+3)$, -1 $(x+1)$, 1 $(x-1)$ and 2 $(x-2)$. Since zero multiplied by any number gives zero, we can try to *multiply these together*. We get:

$$y = (x+3) \times (x+1) \times (x-1) \times (x-2)$$

If you multiply out all the expressions in parentheses you will find that:

$$y = x^4 + x^3 - 7x^2 - x + 6$$

Would you have been able to *go the other way*: go from $x^4+x^3-7x^2-x+6$ to $(x+3)\times(x+1)\times(x-1)\times(x-2)$? It's much harder to do, and it's called *factoring a polynomial*.

The factored expression is useful because it tells you clearly where all the roots of the polynomial are. The normal expression is useful because you can see clearly the coefficients of the terms of each order. In particular, the fact that the x^4 has a positive coefficient (in this case 1) and an even power allows you to say that the curve will have arms that point up on both sides.

Another bit of terminology: the zoomed plot in Figure **??** shows a lot of *structure* near $x=0$, but the unzoomed plot in Figure **??** shows that for large negative and positive values of x you just go up. The behavior of a function when you go to large values of x (or large negative values) is called *asymptotic* behavior, and understanding the asymptotic behavior of a function is a useful and deep area of mathematics.

Note that with simple polynomials, it is relativly easy to graph manually and to find the y for a given x. However with higher order polynomials a plotting program helps us gain insight.

4.5 Inverse functions

FIXME: must write; discuss 1/x and x^-n and x^-(a/b)

Mention *inverse proportionality*.

When one

4.6 Elementary transcendental functions

A function which cannot be written as a polynomial is called a *transcendental function*.

4.6.1 Exponentials

Exponential growth and decay are commonly seen phenomena in daily life. The exponential function is be used to model many real life situations with growth and decay.

Growth

Let us start with a simple problem and a small program to simulated it. Let's say we have a pair of rabbits, and every month they produce two baby rabbits. (NOTE: rabbits do have a one month gestation time, but their litters are actually much bigger – 4 to 12 kits, but we're keeping it simple).

After one generation you have 2 new rabbits (a total of 4). Then those pair up and reproduce and you have 4 new rabbits (a total of 8).

A simple program to demonstrate this is:

Listing 4.6.1: Program which models simple reproduction.

```
#! /usr/bin/env python3

import math

def main():
    n_generations = 10
    n0 = 2                      # initial population
    print('## generation n_rabbits')
    # print('## generation n_rabbits exponential')
    n = n0
    for generation in range(1, 11): # let's do 10 months
        n_new = (n / 2.0) * 2
        n = n + n_new
        print(generation, n)
        # print(generation, n, math.pow(2.0, generation+1))

main()
```

Run this program and notice that the

The general formula for exponential growth or decay is written $A(t) = P_0 e^{r*t}$, where A is the amount at any given t, time, r is the *rate* of growth or decay, and P_0 is the initial amount of whatever is being changed. It is important to remember here that e is simply a number – it is not a variable, and should remain constant in each situation. In this way, it should be treated the same as pi.

Let us explore this function:

$$f(x) = e^x$$

Wait, wait wait: what does this even mean? We have seen what it means to take a number, like 4, and raise it to a power, like 3: $4^3 = 4 \times 4 \times 4 = 64$. But what does it mean to take numbers that are *not* integers and raise them to powers? That's straightforward if the power is an integer. For example: $1.7^3 = 1.7 \times 1.7 \times 1.7 = 4.913$. That's all fine.

But what does it mean to take a number to a power that is not an integer? The topic is too long to go in to in detail, but remember this property you probably learned a long time ago:

$$x^{2+1} = x^2 \times x^1$$
$$x^{m+n} = x^m \times x^n$$

This allows us to define fractional exponents:

$$x^1 = x^{1/2+1/2} = x^{1/2} * x^{1/2}$$

which means that $x^{1/2}$ is that number which, when squared, gives x, which is the very definition of square root! Thus:

$$\begin{aligned} x^{1/2} &= \sqrt{x} \\ x^{1/3} &= \sqrt[3]{x} \\ &\ldots \\ x^{1/n} &= \sqrt[n]{x} \end{aligned}$$

Using the properties of exponents that we know, this allows us to get pretty close to almost all expressions a^b where a and b are real numbers, not just integers or fractions.

The number e is a special number, called *Euler's number*. It's numeric value is about 2.71828, and it comes up in so many places in math that it is as important as :math:pi`.

Let us look at our rabbit reproduction problem and write out an extra number. Change the line that prints the current population to say:

```
print(generation, n, math.pow(2.0, generation+1))
```

Note that you will also need to `import math` at the top of your program.

If you run the program again you will see that the exponential function $2^(generation + 1)$ matches what our simulation of rabbit growth gives.

At some point you will learn that you can convert an exponential in base e to an exponential in base 2, or 10, or any other number. The three important ones are e (occurs naturally in math), 2 (comes up in computer science) and 10 (we have 10 fingers, and our digits are 0 to 9). This allows us to write:

FIXME

Exercise 4.1

Introduce rabbit death. Then think about introducing foxes. Refer to the proposed chapter on predator-prey problems.

Decay

Uranium 235 has a half-life of 703 million years. Let's write a program to simulate how some 5×10^22 atoms of U235 would decay.

Listing 4.6.2: simple-decay.py - a program which models simple decay.

```
#! /usr/bin/env python3

def main():
    n_steps = 10
    ## U235 density is 19.1 g/cm^3
    n0 = 5e22                          # initial population
    print('## time_yr N_U235')
    n = n0
    time = 0
    ## U235 half life 703.8 million years
    while time < 7.03e10:        # 100 half-lifes
        n = (n / 2.0)
        print('%10.4g    %10.4g' % (time, n))
        time += 7.03e8

main()
```

Run the program and discuss what the meaning of half-life is and how it relates to exponential decay.

4.6.2 Trigonometric function

4.7 Gaussian distribution

Gaussian distribution, more simply known as the bell curve or normal distribution, is found in many places in the world around us. When the heights of children of a similar ages are graphed in relation to one another, the result is a bell curve. Named for its distinctive, bell like shape, the bell curve can be easily graphed with knowledge of a relatively simple equation. It is :(equation-formatting?) where () is the mean, and () is the standard deviation.

Now, of course we could calculate these values manaually, but why would we?Use (code) to generate a random set of numbers. Start out with about 20.

Now, use () to find the sum of these numbers. Check your answer logically- does this seem accurate? Now, divide this by the number of values you used.

This works, but it would be easy to cut out the middle step- use (code) to determine the mean with no possibility for user error.

Next, you can find the standard deviation- use (code).

Now, you're all set up to plug these numbers into the equation listed above. But this seems time consuming, and programming is all about finding ways to minimize repetitive tasks. Use (code) to find the standard deviation of your data set.

Try it with data for the heights of your school, or for some other large data set that you find interesting.

Next, plot this data using (code).

Try manipulating the graph using the above techniques.

CHAPTER

FIVE

GROWTH – CHECKED AND NOT

[status: just starting]

5.1 Motivation, prerequisites, plan

5.1.1 Motivation

I am preparing this mini-course on 2020-03-12 as New Mexico has just entered a state of emergency over the coronavirus epidemic. It seems topical and a nice opportunity to introduce our students to some aspects of the mathematical equations that describe growth of species.

5.1.2 Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**
- The "A tour of functions" mini-course in Section **??**

5.1.3 Plan

There is an attractive and accessible video at:

https://www.youtube.com/watch?v=Kas0tIxDvrg

and:

https://www.youtube.com/watch?v=PUwmA3Q0_OE

which students can watch before the mini-course.

Then we go on to show how difference equations can lead to exponential growth for an unchecked population.

Then on to the logistic equation and how exponential growth actually works in the real world.

Finally a discussion of predator-prey models.

5.2 Pure exponential growth

Discus Malthus, what “malthusian” means.

Then we review what the exponential function looks like compared to this progression.

At the python3 interpreter type:

```
P0 = 10
rate = 1.2

P = P0
print('## generation population')
for generation in range(100):
    P = P + P * (rate-1)
    print(generation, '    ', P)
```

Now put that code in a file and run the program, saving its output so that we can plot it.

Now compare that to what would have come out of a straight exponential function $f(x) = P0 \times e^{rt}$. Can you make the rate and initial population correspond between the difference equation?

https://en.wikipedia.org/wiki/Population_growth

$dP/dt = rP(1 - P/K)$

or

$df(x)/dx = f(x)(1 - f(x))$

Solution is:

f(x) = exp(x) / (exp(x) + C)

discuss the exponential phase initially, then the cooling down phase. talk about fidget spinner fads.

P = P0

K = 3.7 is the “carrying capacity”

https://en.wikipedia.org/wiki/Logistic_function

limit t -> infinity P(t) = K

From wikipedia: “In ecology, species are sometimes referred to as r {displaystyle r} r-strategist or K {displaystyle K} K-strategist depending upon the selective processes that have shaped their life history strategies.”

5.3 Checked growth

5.3.1 Checked by lack of resources

5.3.2 Checked by competition with other species

5.4 Simple predator-prey interactions

5.4.1 The Lotka-Volterra equations

The relationship between populations of predators and prey can be expressed by the Lotka-Volterra equations, a set of paired differential equations that approximate the interaction between predator and prey populations over many generations.

The Lotka-Volterra equations are commonly expressed as two paired differential equations, where h is the population of prey and p is the population of predators. Notice the parralels between the first and second terms of each equation.

$$\frac{dh}{dt} = ah - bhp$$
$$\frac{dp}{dt} = -cp + dhp$$

a, b, c, and d are variables which dictate the nature of the interaction. In the case of the prey, a represents the growth of the prey population. Since it is multiplied by the current prey population h, the population growth of the prey is exponential at its base, excepting the second term. Conversely, c represents the exponential decay of the predator population in the absence of any prey to eat.

The second term of each equation reflects how often predators and prey encounter each other and the effect these encounters have on their respective populations. Before we look into what hp is doing in the second term of both differentials, it's important to understand what the general purpose of the second term is in each case.

For the prey, the exponential population growth from the first term is checked by predation. In the case of the predator population, the exponential decay they experience without external input is curbed by the same force of predation. This is the basis of why both differentials include the term hp.

However, this makes more sense when we look more closely at what hp means in the context of this equation. By multiplying the populations of the predators and prey, we can find the dimension of the populations. On the surface this seems like a nonsensical calculation, but it is integral to the function of the Lotka Volterra equations.

If there are many more prey than predators (a large dimension), it is easy for predators to find something to eat. Similarly, we also notice a large dimension if there are mant more predators than prey. In this case, it is easy for prey to get caught by a predator. The dimension of the populations reflects the number of interactions we can expect between predators and prey, but it doesn't necessarily tell us about which side benefits most.

We can only get this information by looking back at the differential equations. b and d are variables which affect how strongly the dimension of the populations effect each population.

CHAPTER

SIX

ADVANCED PLOTTING

[status: content-partly-written]

Motivation

A quick plot is OK when you are exploring a data set or a function. But when you present your results to others you need to prepare the plots much more carefully so that they give the information to someone who does not know all the background you do.

Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**.
- The "Intermediate plotting" mini-course in Section **??**.

Plan

We will go through some topics in plotting that might come up when you need to prepare a *polished* plot for publication, or an animation for the accompanying materials in a publication.

This chapter will have a collection of examples that you might adapt to use in your papers or supporting web materials. They will all be based on the physical system of the *harmonic oscillator*, in this case a mass bouncing on a spring. (The harmonic oscillator equations are discussed and solved in more detail in Section **??**.)

After showing simple oscillator line plots we will work through examples of adding touches that convey information about the axes and lines, as well as showing example captions for plots.

Then we will add another dimension: line plots that depend on a parameter can be viewed in interesting ways that give insight.

It's hard to go beyond a surface plot, but there are some techniques to visualize more complex systems. Techniques that do this are volume rendering, isosurfaces and animations, and we will give examples of these.

6.1 Our setup

The same simple harmonic oscillator equations can describe very many physical systems. Ours will be a mass bouncing sideways on a spring as shown in Figure **??**.

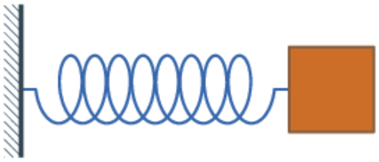


Figure 6.1.1: A mass bouncing on a spring. The zero position on the x axis is when the spring is at rest and does not push or pull the mass.

When we teach this in introductory physics classes we assume that there is no friction as the mass bounces to the right and left. We call that the *simple harmonic oscillator* (SHO). If there is friction it is a *damped harmonic oscillator*.

The spring has a *spring constant* k which represents how stiff it is. The force that returns the spring toward the rest position after you pluck it a distance x is $F = -k \times x$. The mass is m.

We are interested in the *position* of this mass as a function of *time*. The equation for this is:

$$x(t) = A\cos(\omega t) \tag{6.1.1}$$

This equation has two variables x and t, and two constants, A and ω.

A is the *amplitude* of the oscillation: the maximum displacement x will ever reach, both to the right and to the left. The angular frequency of the cosine wave is given by $\omega = \sqrt{\frac{k}{m}}$. A few simple things we can say about this formula are:

- The formula comes from solving Newton's law: $F = ma$ where the force is $-kx$, so we have $a + \frac{k}{m}x = 0$. Since the accelartion a is the second derivative of the position $\frac{d^2x}{dt^2}$ or $\ddot{x}$, so we have

  $$\frac{d^2x}{dt^2} + \frac{k}{m}x = 0$$

  which is a differential equation which is solved by equation (**??**). I don't discuss how to solve differential equations here (see Section **??**) since we are only interested in the plotting of solutions, so you can ignore that: we will focus on plotting the solution. I only mention the equation it comes from so you can start getting excited about the amazing world of mathematical physics that lies ahead of you.

  Note: For teachers: this mention of of differential equations should be very rapid, or you could skip it altogether. Mentioning it gives a vision of depth, but you need to slip it in casually and without spending too much time on it, or it can be daunting for some students. You could say "you will see this toward the end of high school or in college; for now just look at it and let's move on to the solution."

- If you pluck the spring to the right and then let it go, the amplitude A is how far you plucked it initially.
- The frequency $f = \omega/(2\pi)$ is greater with a stiffer spring, and smaller with a larger mass. The period is the inverse of the frequency: $T = 1/f = 2\pi/\omega$.

6.2 A first line plot

Let us now jump in to generating data for this system and plot it. A very simple program will generate the trajectory. Let us pick amplitude, spring constant and mass: $A = 3, k = 2, m = 1$, and we have $x(t) = 3\cos(2t)$. This program will generate the data:

Listing 6.2.1: generate-SHO.py - generate data for the simple harmonic oscillator solution.

```
#! /usr/bin/env python3

import math

A = 3.0
k = 2.0
m = 1.0
w = math.sqrt(k/m)

for i in range(140):
    t = i/10
    x = A * math.cos(w * t)
    print(t, '     ', x)
```

Run this program with

```
chmod +x generate-SHO.py
./generate-SHO.py > SHO.dat
```

You can then plot it with something like:

```
gnuplot> plot 'SHO.dat' using 1:2 with linespoints
```

What you see is a plot of $x(t)$ versus t.

This plot gives you some insight: it shows you that if you pluck the spring to the right by 3 units it will bounce back and forth regularly, with an oscillation period of about 4 and a half units of time (more precisely $T = 2\pi/\sqrt{2} \approx 4.44$.

This plot is OK for you to get quick insight into the output of your `generate-SHO.py` program.

But you would certainly not want to use it to present your results to someone else because of several problems:

- There is no title in the plot.
- The x (time) and y (x(t)) axes are not labeled.
- The line has a very generic legend. It probably says something like `'SHO.dat' using 1:2`, which is not informative.

In some situations this default plot might have other problems, like:

- The + signs for points might not be what you want (maybe a filled circle, or a hollow box are better).
- You might want a bit of padding on the y axis since the plot touches the top.
- You might want to change the *aspect ratio* of the plot. The default is a rectangular overall shape, but sometimes you might want a different shape of rectangle, and sometimes you might want a square. Examples of a square aspect ratio are in Section **??**.
- There may be too many numbers on the x or y axes, so that they crowd in to each other.

6.3 Polishing line plots

Let us address the first two issues. We can choose labels for the x and y axes like this:

```
gnuplot> set xlabel 'time t (seconds)'
gnuplot> set ylabel 'spring displacement x(t) (cm)'
gnuplot> plot 'SHO.dat' using 1:2 with linespoints
```

With these instructions I have also communicated the units of measure I am using for these plots. Without units the plots are junk.

As for the legend for the line, we use the *title* directive in gnuplot's plot command. Something like:

```
gnuplot> set xlabel 'time t (seconds)'
gnuplot> set ylabel 'spring displacement x(t) (cm)'
gnuplot> plot 'SHO.dat' using 1:2 with linespoints title 'simple harmonic oscillator'
```

We can also give the plot an overall title. This is not too interesting when we have a single line drawing, but if we have several lines then an overall title is useful.

```
gnuplot> set title 'Position versus time for harmonic oscillators'
gnuplot> set xlabel 'time t (seconds)'
gnuplot> set ylabel 'spring displacement x(t) (cm)'
gnuplot> plot 'SHO.dat' using 1:2 with linespoints title 'simple harmonic oscillator'
```

Gnuplot has very many options you can manipulate, and there is a vast collection of demos at http://gnuplot.sourceforge.net/demo/ – it is quite rewarding to go through all the demos and adapt them to the things you might need to plot.

6.4 Increasing the challenge: parametrized solutions

The simple harmonic oscillator is not a terribly rich system, so we will add a bit more to it by looking at the *damped harmonic oscillator*. The equation for this is:

$$\frac{d^2x}{dt^2} + c\frac{dx}{dt} + \frac{k}{m}x = 0$$

where c is a constant which expresses how much the system is damped by friction.

The solution is:

$$x(t) = A \times e^{-ct/(2m)} \times \cos(\omega t) \tag{6.4.1}$$

where the frequency ω is:

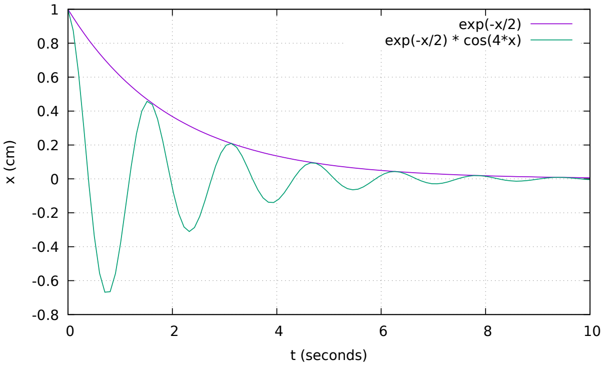
$$\omega = \sqrt{k/m - c^2/(4m^2)} \tag{6.4.2}$$

Let's do a quick function plot to try to underestand what a negative exponential times a cosine function looks like. In gnuplot you can type these instructions:

Listing 6.4.1: Gnuplot instructions for a sample damped cosine function.

```
set grid
set xlabel 't (seconds)'
set ylabel 'x (cm)'
plot [0:10] exp(-x/2), \
     exp(-x/2) * cos(4*x)
```

The result should look like:



Now that we have an idea of the shape of a damped cosine curve, let us look briefly at equations (**??**) and (**??**) and imagine in our minds how they might behave.

Overall the shape of the damped oscillator solution should look like `fig-sample-damped-cos`

The exponential factor $e^{-c/(2m)}$ is what makes the oscillations get smaller, and if the friction c is big then it will damp much faster, while if the mass is big it will damp less (more inertia).

The cosine factor $\cos(\omega t)$ means that you have oscillations (at least until it has damped a lot), and equation (**??**) tells us that the friction has made the frequency of the oscillations somewhat smaller than the natural frequency $\sqrt{k/m}$ of the undamped oscillator.

All of this depends on the friction constant c.

But how do we visualize this curve for different values of c? Let us start by writing a program which prints out several columns for $x(t, c)$ with a few different values of c.

Listing 6.4.2: damped-oscillator-columns.py - generate several columns of data with different damping constants.

```
#! /usr/bin/env python3

import math

A = 3.0
k = 2.5
m = 1.0

damping_constants = [0, 0.5, 1.0, 1.5, 2.0]
print('## time', end="")
for c in damping_constants:
    print('        c_%f' % c, end="")
print()
```

(continues on next page)

(continued from previous page)

```
for i in range(300):
    t = i/10
    print(t, end="")
    for c in damping_constants:
        w = math.sqrt(k/m - c*c/(4*m*m))
        x = A * math.exp(- t * c*c/(2*m)) * math.cos(w * t)
        print('      ', x, end="")
    print()
```

You can run this with

```
chmod +x damped-oscillator-columns.py
./damped-oscillator-columns.py > damped-oscillator-columns.dat
```

and then plot it with:

Listing 6.4.3: Gnuplot instructions to plot the output columns of damped-oscillator-columns.py

```
set grid
set title 'Damped harmonic oscillator with various damping factors'
set xlabel 'time t (seconds)'
set ylabel 'spring displacement x(t) (cm)'
plot 'damped-oscillator-columns.dat' using 1:2 with lines title 'c=0.0' lw 7 , \
     'damped-oscillator-columns.dat' using 1:3 with lines title 'c=0.5' lw 7 , \
     'damped-oscillator-columns.dat' using 1:4 with lines title 'c=1.0' lw 7 , \
     'damped-oscillator-columns.dat' using 1:5 with lines title 'c=1.5' lw 7 , \
     'damped-oscillator-columns.dat' using 1:6 with lines title 'c=2.0' lw 7
```

The result should look like:

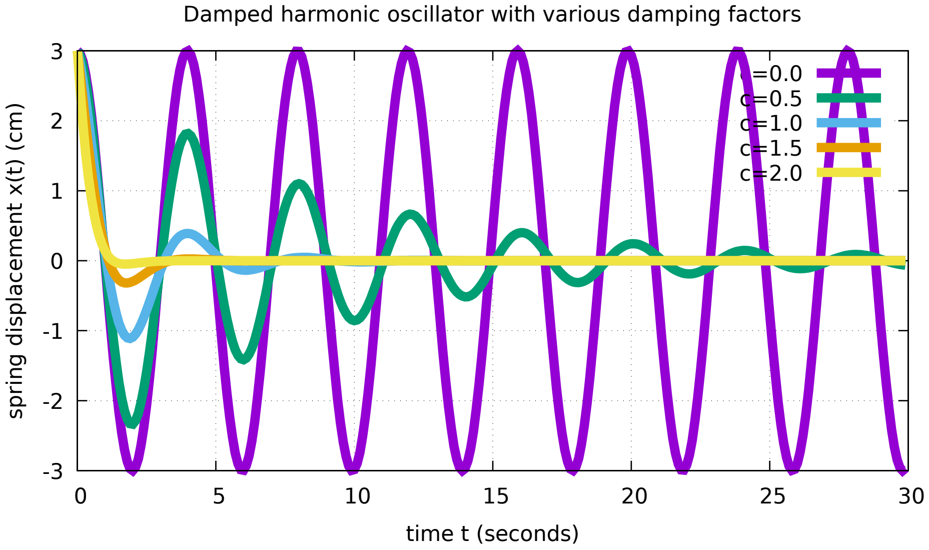


Figure 6.4.1: Damped harmonic oscillator with separate line plots for different values of the damping constant c.

Figure **??** gives us some information if we pick through it. The legend tells us which colored line has $c = 0.0$, and that line should look like there is no damping, which is correct.

Then we see the line with $c = 0.5$ and we notice that (a) the peaks of the oscillation get smaller (the damping), (b) if you follow it further out you see that the peaks don't line up with the peaks of the $c = 0$ curve (the frequency is a bit lower than the undamped curve, which validates equation (**??**)).

Then we see that the lines with more damping (c = 1.0, 1.5, 2.0) damp more quickly.

But we had to really pick through this plot and strain our eyes and patience to see those effects. Let us see if we can make the information about the c parameter come out more clearly.

6.5 Adding a dimension: surface plots

Listing 6.5.1: damped-oscillator-surface.py - generate damped harmonic oscillator data for different values of the damping constant c. Sections of data are separated by a blank line, which is what gnuplot needs to make a surface plot.

```
#! /usr/bin/env python3

import math

A = 3.0
k = 2.5
m = 1.0

print('## time  damping_c  x')

for ci in range(0, 60):          # loop on damping constants
    c = ci/40.0
    for ti in range(300):        # loop on time
        t = ti/10
        w = math.sqrt(k/m - c*c/(4*m*m))
        x = A * math.exp(- t * c*c/(2*m)) * math.cos(w * t)
        print(t, '   ', c, '    ', x)
    print()                     # blank line between damping constants
```

We can run this program with

```
chmod +x damped-oscillator-surface.py
./damped-oscillator-surface.py > damped-oscillator-surface.dat
```

Listing 6.5.2: damped-oscillator-surface.gp - make a surface plot of the damped oscillator displacement as a function of both time and the damping constant c.

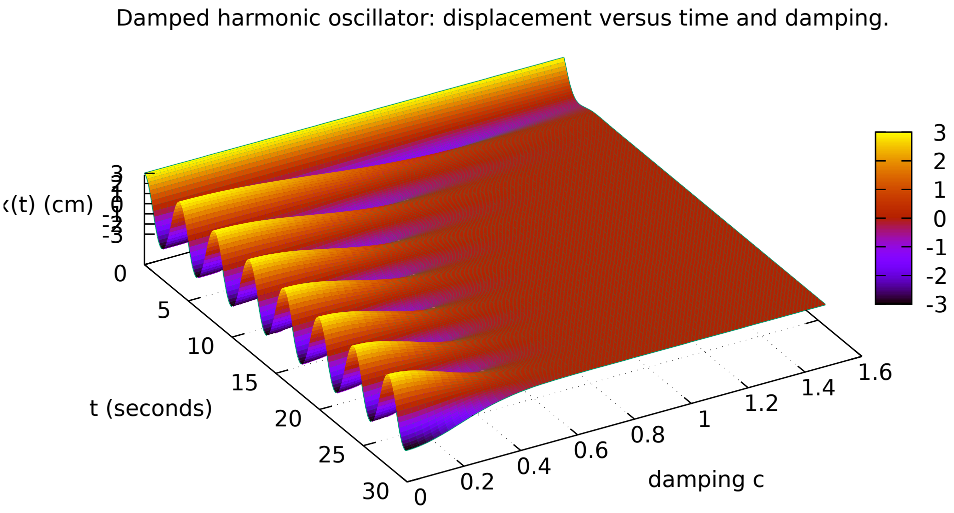
```
set grid
set title 'Damped harmonic oscillator: displacement versus time and damping.'
set xlabel 't (seconds)'
set ylabel 'damping c'
set zlabel 'x(t) (cm)'
set pm3d
set hidden3d
```

(continues on next page)

(continued from previous page)

```
set view 20, 60     ## set the viewpoint
splot 'damped-oscillator-surface.dat' using 1:2:3 with lines title ''
```

The surface plot looks like:



The main difference between `fig-damped-oscillator-surface` and the figure with several lines: with the surface you can see the behavior of x versus t for very many values of c without having to scrunch your eyes. With a single glance you can see the effect of the damping constant: a downslope in the oscillatory behavior.

6.6 Adding a dimension: color

6.6.1 Spetrograms

If you want to represent the information about a voice, or music, or an animal call, or a radio signal.

Think carefully of what kind of information is in this type of signal. You will need to represent the *intensity* of the signal (for example how loud the music is or how strong a signal is) for each frequency, and all of this will depend on time.

One tool to visualize such signals is the *spectrogram*. Let's start with picture:

In Figure **??** we see that the x axis is time, the y axis is frequency, and the loudness of the sound (in decibels) is represented by the color: black and dark red are more quiet, bright yellow and white are louder.

What you see visually is that the various features of the sound are seen in features of the spetrogram: the chirps are the inverted V shapes, the clicks are vertical lines, and the harmonizing is seen in the horizontal striations.

Once you get used to reading spectrograms you can quickly get a feeling for not just the volume of sound as a function of time, but also for *in which frequencies* those louder sounds occur.

Thus the use of color helps us get a quick-look view of the entire picture, and it helps us find distinguishing features in the data.

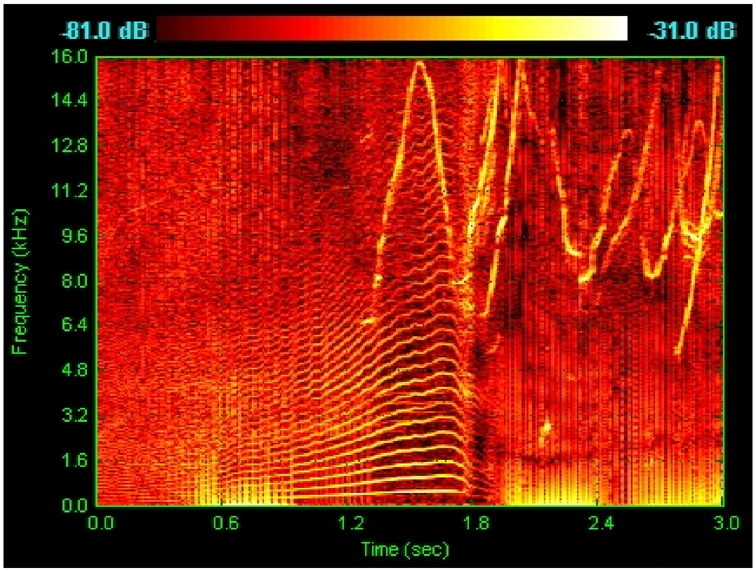


Figure 6.6.1: Spetrogram of a dolphin's vocalizations. You can see chirps, clicks and harmonizing. (From Wikipedia)

6.6.2 Spectrograms for standard acoustic files

Let us download a brief violin sample, specificially a single F note.

```
wget --continue http://freesound.org/data/previews/153/153595_2626346-lq.mp3 -O violin-F.
↪mp3
```

This gives us the file `violin-F.mp3`. It's hard to load this file in as data, but we can do a couple of conversions with the programs *ffmpeg* (which manipulates audio and video formats) and *sox*, which manipulates some audio formats and can turn audio files into human-readable columns of data.

```
ffmpeg -i violin-F.mp3 violin-F.aif
sox violin-F.aif violin-F.dat
```

We can look at `violin-F.dat` and see that it has three columns of data. The first is time, the second is the left stereo channel, and the third is the right stereo channel.

For simplicity we will analyze the left stereo channel. We use this

Let us record our own signal so that we can experiment.

Type

```
rec myvoice.dat
```

then speak in to it for no more than two seconds and hit control-C to abort. You now have a file with time and one or two amplitude channels. You can look at it and plot it.

You can also record yourself playing a musical instrument, or clapping.

To play the sample back you can type:

```
play myvoice.dat
```

and you should hear back your brief audio sample.

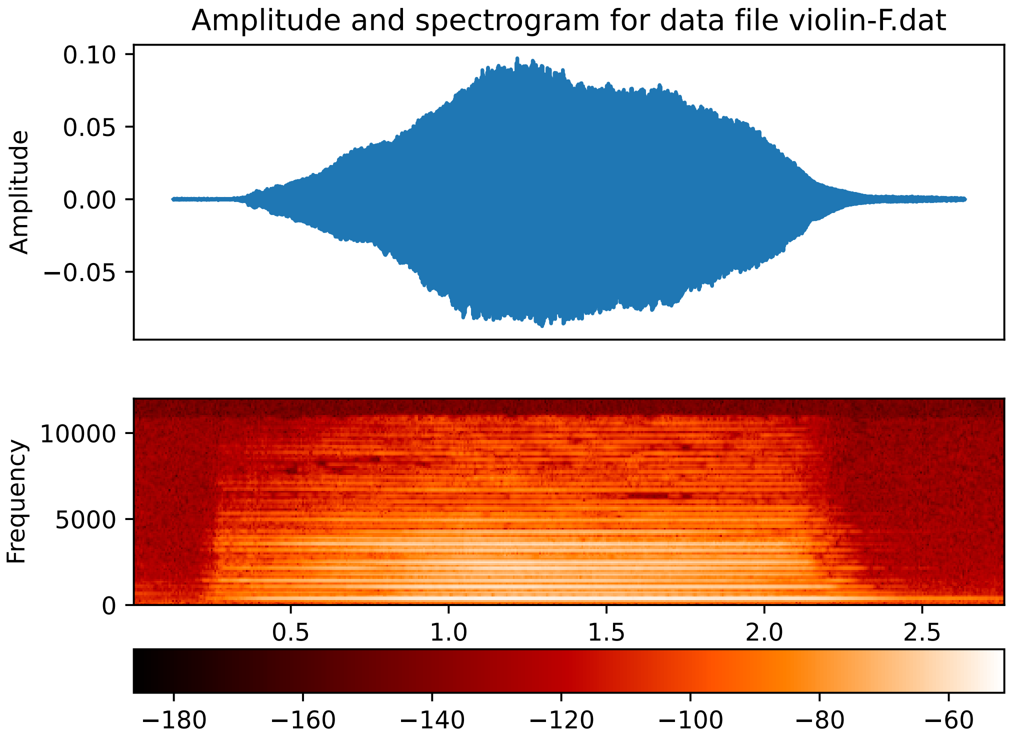


Figure 6.6.2: Spetrogram of a violin playing an F (Fa) note. Note the fundamental frequency at the bottom, and the many harmonics above that.

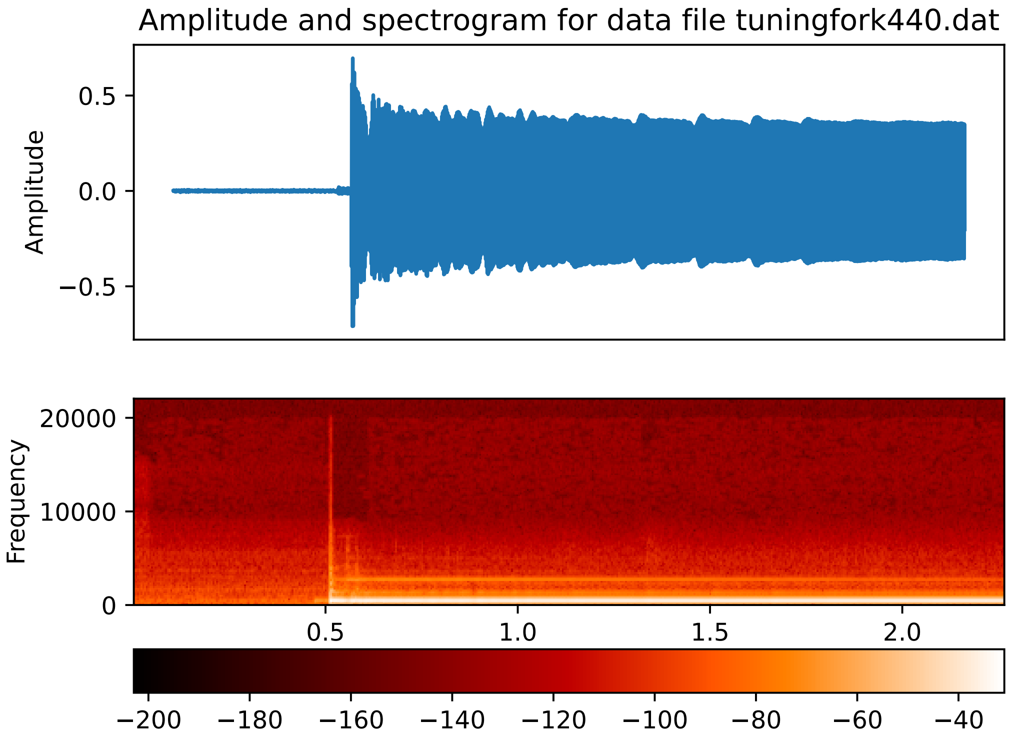


Figure 6.6.3: Spetrogram of the note of a tuning fork. This produces a 440 Hz A (La) sound. Note that there are almost no harmonics: the tuning fork is designed to put out a sound close to a pure sin wave.

But remember: we are scientists, not just users of gadgets, so when we think of an audio file we think of it as a *data*. You can look at the file with

```
less myvoice.dat
```

and you will see three columns of numbers. The first is time, the others are the amplitude of the sound in the left and right stereo channels.

Listing 6.6.1: music2spectrogram.py - Takes an audio file and makes a spectrogram of it. You can also download it at this link: `music2spectrogram.py`

```
#! /usr/bin/env python3

import sys
import numpy as np
import matplotlib.pyplot as plt
import pylab

def main():
    if not len(sys.argv) in (2, 3):
        print('error: please give an audio file argument')
        sys.exit(1)
    infile = sys.argv[1]
    ## find the sampling rate (in samples/sec) from the input file
    Fs = get_sample_rate(infile)
    outfile = None
    if len(sys.argv) == 3:
        outfile = sys.argv[2]
    time = np.loadtxt(sys.argv[1], comments=';', usecols=(0,))
    left_channel = np.loadtxt(sys.argv[1], comments=';', usecols=(1,))
    ## ignore the right channel (if there is one)

    ax_plot = plt.subplot(211)
    ax_plot.set_title('Amplitude and spectrogram for data file %s' % infile)
    plt.plot(time, left_channel)
    plt.ylabel('Amplitude')
    ax_plot.set_xticks([])

    ax_spec = plt.subplot(212)
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    Pxx, feqs, bins, im = plt.specgram(left_channel, Fs=Fs, cmap=pylab.cm.gist_heat)
    cbar = plt.colorbar(orientation='horizontal') # show the color bar information
    if outfile:
        plt.savefig(outfile)
    else:
        plt.show()

def get_sample_rate(fname):
    """Read the first line in the file and extract the sample rate
    from it.  These ascii sound files are like that: two lines of
    "metadata", of which the first has the sampling rate and the
    second lists how many channels there are.  This routine reads that
```

(continues on next page)

(continued from previous page)

```
    first line and returns the number.  For compact disc music, for
    example, it would be 44100 Hz.
    """
    with open(fname, 'r') as f:
        line = f.readline()     # get the first line
    assert(line[:len('; Sample Rate')] == '; Sample Rate')
    Fs = int(line[(1+len('; Sample Rate')):])
    print('sample rate:', Fs)
    return Fs


main()
```

We can run this program with

```
chmod +x music2spectrogram.py
./music2spectrogam.py violin-F.dat
## or to save it to a file:
./music2spectrogram.py violin-F.dat violin-F_spectrogram.png
```

6.6.3 Ideas for further exploration

Earthquakes : color is magnitude

6.7 Loops in gnuplot

Look at examples from here:

http://gnuplot.sourceforge.net/demo/bivariat.html

in particular the fourier series.

6.8 Adding a dimension: animation

6.8.1 Animation in gnuplot

[FIXME: incomplete]

http://personalpages.to.infn.it/~mignone/Algoritmi_Numerici/gnuplot.pdf

also:

http://www.gnuplotting.org/tag/do/

6.8.2 Animation in matplotlib

simple-animated-plot.py

Listing 6.8.1: simple-animated-plot.py – . . .

```
#! /usr/bin/env python3

## example taken from https://stackoverflow.com/a/42738014

import matplotlib.pyplot as plt
import numpy as np

plt.ion()
fig, ax = plt.subplots()
x, y = [],[]
sc = ax.scatter(x,y)
plt.xlim(0,10)
plt.ylim(0,10)

plt.draw()
for i in range(1000):
    x.append(np.random.rand(1)*10)
    y.append(np.random.rand(1)*10)
    sc.set_offsets(np.c_[x,y])
    fig.canvas.draw_idle()
    plt.pause(0.1)

plt.waitforbuttonpress()
```

simple_anim.py

Listing 6.8.2: simple_anim.py – . . .

```
#! /usr/bin/env python3
"""
==================
Animated line plot
==================

"""

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

fig, ax = plt.subplots()

x = np.arange(0, 2*np.pi, 0.01)
line, = ax.plot(x, np.sin(x))


def init():  # only required for blitting to give a clean slate.
    line.set_ydata([np.nan] * len(x))
```

(continues on next page)

(continued from previous page)

```
    return line,


def animate(i):
    line.set_ydata(np.sin(x + i / 100))  # update the data.
    return line,


ani = animation.FuncAnimation(
    fig, animate, init_func=init, interval=2, blit=True, save_count=50)

# To save the animation, use e.g.
#
# ani.save("movie.mp4")
#
# or
#
# from matplotlib.animation import FFMpegWriter
# writer = FFMpegWriter(fps=15, metadata=dict(artist='Me'), bitrate=1800)
# ani.save("movie.mp4", writer=writer)

plt.show()
```

using matplotlib's `FuncAnimation`:

```
import matplotlib.animation as animation
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()

def updatefig(i):
    fig.clear()
    p = plt.plot(np.random.random(100))
    plt.draw()

anim = animation.FuncAnimation(fig, updatefig, 100)
anim.save("/tmp/test.mp4", fps=24)
```

And from the matplotlib docs:

https://matplotlib.org/gallery/animation/simple_anim.html

6.9 Further reading

See also:

clean plots:

http://triclinic.org/2015/04/publication-quality-plots-with-gnuplot/

http://gnuplot.sourceforge.net/demo/

https://www.jstor.org/stable/pdf/2683253.pdf

more about sox and audio formats:

http://sox.sourceforge.net/sox.html

https://newt.phys.unsw.edu.au/jw/musical-sounds-musical-instruments.html

object oriented approach in matplotlib:

https://python4astronomers.github.io/plotting/advanced.html

more on matplotlib animation

https://nickcharlton.net/posts/drawing-animating-shapes-matplotlib.html

cool matplotlib resources:

https://realpython.com/python-matplotlib-guide/

https://python4astronomers.github.io/plotting/advanced.html

CHAPTER

SEVEN

FITTING FUNCTIONS TO DATA

[status: content_complete_needs_review]

7.1 Motivation, Prerequisites, Plan

In Section **??** we looked at how to take a function and represent it as points and lines in a plot. Here we do the opposite: if we are given a collection of (x, y) points we try to find *what kind of function might have generated those points*.

There are so many types of functions that there is some artistry involved in picking which kind of function to fit to a set of data.

In many cases we will want to fit a straight line to our data. Sometimes it will be a polynomial. Our intuition on what functions to try to fit can come from two sources: (a) looking at the data visually ("hey! that looks like a parabola!"), and (b) having some intuition about the *process* that underlies that data ("should be an exponential decay; let's try fitting an exponential function!")

Terminology: the techniques we use here are sometimes referred to as regression, linear regression, fitting, curve fitting, line fitting, least squares fitting, …

7.2 Examples to get started

Let us start by visualizing some data sets. We'll start with the dataset we had for age, height and weight of the !Kung people of the Kalahari desert. We downloaded and analyzed this data set back in Section **??**, but there we were looking at height and weight historgrams. Here we look at something simpler: height versus age.

In this example we will grab the data set,

```
$ wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/Howell1.csv
$ gnuplot
gnuplot> set datafile separator ";"
gnuplot> plot 'Howell1.csv' using 3:1 with points
```

You can carry it all out with the following instructions:

Listing 7.2.1: Instructions to plot the height vs. age for the !Kung.

```
##REQUIRED_FILE: Howell1.csv
##PRE_EXEC: wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/
↪Howell1.csv
set datafile separator ";"
```

(continues on next page)

(continued from previous page)

```
set grid
set title 'height versus age for the !Kung people'
set xlabel 'age (years)'
set ylabel 'height (cm)'
plot 'Howell1.csv' using 3:1 with points
```

and you should get the figure in Figure **??**.

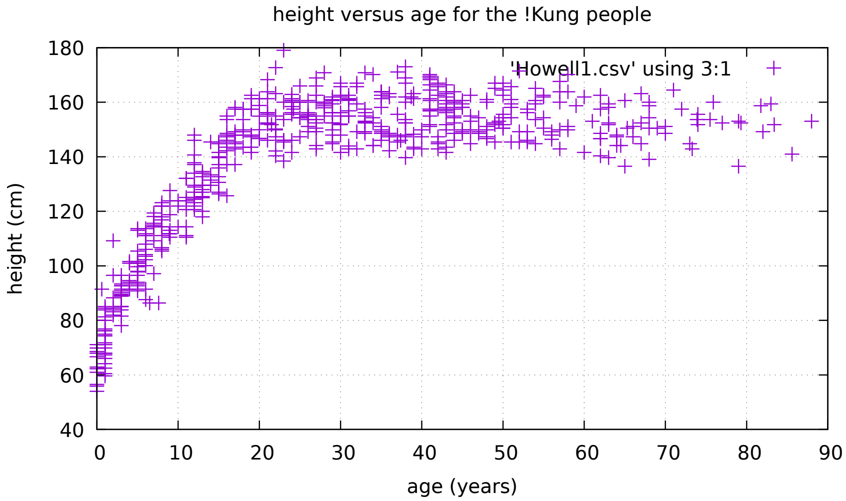


Figure 7.2.1: Height vs. age for the !Kung people of the Kalahari desert. Note that we have two separate behaviors of the height data: one for before the age of 18 (rapid growth), the other for after the age of 20 (mostly norizontal).

We see this drastic difference between lower and higher age. One thing that comes to mind is that the mechanisms that cause this are obvious biological ones: we grow fast when we are young, then we stop growing taller.

The rising height part of the graph (up to age 18) seems to be reasonably close to a straight line, as is the top part (from 20 years on). The entire plot does not at all look like a straight line.

Let us look at those separate areas in separate plots:

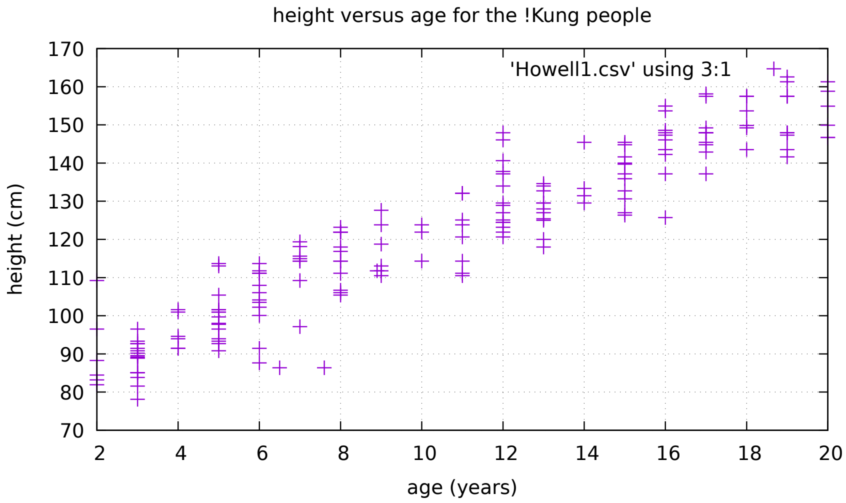


Figure 7.2.2: Zooming in on the age 2-18 region of the height plot.

Listing 7.2.2: Zooming in on the straight line between ages 2 and 18.

```
##REQUIRED_FILE: Howell1.csv
##PRE_EXEC: wget https://raw.githubusercontent.com/rmcelreath/rethinking/master/data/
↪Howell1.csv
set datafile separator ";"
set grid
set title 'height versus age for the !Kung people'
set xlabel 'age (years)'
set ylabel 'height (cm)'
plot [2:20] 'Howell1.csv' using 3:1 with points
```

and you should get the figure in Figure **??**.

7.3 Straight line fits

7.3.1 Our goal

Let us state a bold goal: can we find a straight line that seems to fit the points in the rising area of the height plot of the !Kung people (Figure **??**)?

Remember from Section **??** that a straight line looks like $y = mx + b$ is described by two parameters: the *slope* m and the *y intercept* b, so another way to phrase this goal is:

> Given a collection of points (x_i, y_i), find the slope and intercept for the line $y = mx + b$ which most closely fits the points.

You could do this visually: print it, take a ruler, place it so that it runs through the points in the plot, and draw a line. The result is not really optimal, so we look for better techniques.

7.3.2 Stepping back: just two points

Let us start with just two points, one for a 3-year-old and the other for a 17-year-old. From the data file we can pick the (age, height) pairs: (3, 96.52) and (17, 142.875).

In middle school math we learn how to find the line that goes through these points. We write out the equation $y = mx + b$ using the specific (x, y):

$$96.52 = m \times 3 + b \tag{7.3.1}$$

Note: This is just a first stab at it! We do not know if we chose those two points well – in fact it seems from this picture that the 17-year-old we picked was shorter than average, which would skew the results. The 3-year-old is also taller than average. This example was just to get going so that we can talk about line fits. In Section **??** we will do proper line fitting.

7.3.3 Let's plot that line with our data

Now that we have a line fit we should feel really excited about plotting it together with our data, so that we can get some visual satisfaction. I always encourage people to do this right away.

Let us take the plotting instructions in Listing **??** and add to it the plotting of our fitted line:

Listing 7.3.1: Height vs. age and a line fit for ages 2-18.

```
##REQUIRED_FILE: Howell1.csv
##PRE_EXEC: wget --continue https://raw.githubusercontent.com/rmcelreath/rethinking/
↪master/data/Howell1.csv
set datafile separator ";"
set grid
set key right bottom
set title 'height versus age for the !Kung people'
set xlabel 'age (years)'
set ylabel 'height (cm)'
plot [0:26] 'Howell1.csv' using 3:1 with points, \
     [2:18] 3.311 * x + 86.5867 lw 6 title "two point fit"
```

and you should get Figure **??**.

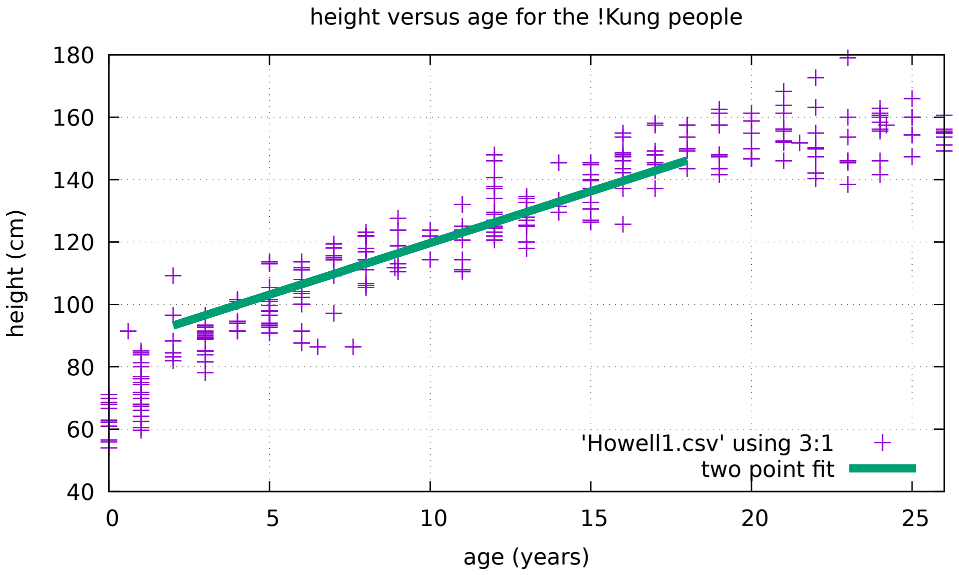


Figure 7.3.1: Height vs. age for the !Kung people of the Kalahari desert, with a line fit for ages 2-18.

7.3.4 Physical interpretation of the line fit

One realy cool thing we learn about fitting functions to data is that those functions have a *physical interpretation*! This gets scientists really excited, so I will mention it before we move on to a better procedure for fitting curves.

If you look closely at the *units of measure* in our scatter plot you see that we are plotting *height* (a measure of length) versus *age* (a measure of time). This means that the slope m is measured in units of length divided by time, in this case in centimeters per year. So the slope we found of 3.311 should really be reported as 3.311 centimeters/year, and it tells us how much children grow per year between the ages of 2 and 18.

Now let us look at the intercept b (in our case 86.5867). This number tels us what the value of y (height) is at time zero (birth). This would seem to imply that babies are born some 87cm tall.

Important: But wait! Babies are not born 87cm tall. They are born around 50cm tall. Then why did our straight line fit report that height at birth?? This is explained by remembering the original plot: in Figure **??** we see that the height forms a straight line between the ages of 2 and 18, but *not* for children less than 2. Those children grow much faster, so the straight line is not a valid approximation for infants. This means that, sadly, the intercept b does not give a gratifying physical interpretation in this case.

7.4 Proper line fitting

We’ve seen how to pick two points from our data set and make a line that goes through them, but this approach has some real problems: if I had chosen a very tall 3-year-old and a very short 17-year-old, then the slope would have been much smaller. I could have also skewed it the other way by picking a very short 3-year-old and a very tall 17-year-old, which would have given a much steeper slope. [FIXME: write exercise to do all of this]

So what is a good objective way of finding the “best fit”? This is much debated, and one get get quite subtle and make a real profession of the line fitting business, but for our purposes we will use the most comon approach. It is called “least squares fitting”.

Look at Figure **??**:

You see four points and an example of a line that *tries* to go through those four points. How well did it succeed?

Look at those green lines in the figure: they show how far the point is from the line, and they give a measure of the error in your fit. If you had chosen a radically different blue line, then the sum of all those green lines might be quite big, and you would have a poor fit.

The idea behind “least squares” is that you want to *minimize* the sum of the *squares* of those errors.

There is a lot going on here. You might immediately ask at least two questions: (a) why the sum of the *squares* of the errors? (b) how do you minimize?

For now I will give a very brief answer to the first question: squares are often considered “nicer” functions than just taking the distance itself. You can see this with this plot:

```
$ gnuplot
gnuplot> plot [-3:3] x*x
gnuplot> replot abs(x)
```

both the lines tend to grow bigger when your error is bigger, but the smooth one is more suited to various mathematical techniques.

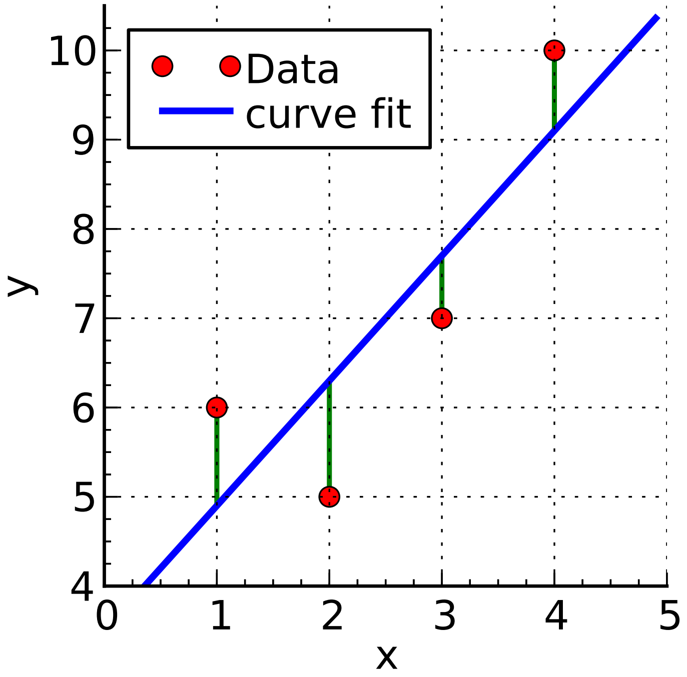


Figure 7.4.1: From wikipedia, https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics) A plot of the data points (in red), the least squares line of best fit (in blue), and the residuals (in green).

As for the second question: we will minimize the sum of the squares of the errors using techniques from calculus. I don't describe them here, but I will show a cool pair of equations. First wrap your head around this expression:

$$E = \sum_{i=1}^{N} (mx_i + b - y_i)^2 \tag{7.4.1}$$

I like formulae like this one because that capital greek sigma letter is visually appealing, but what does it all mean?

First note that in math we use the symbol $\sum_{i=1}^{N}$ to mean "take a sum of all these terms, where the letter i will be replaced in turn by 1, 2, 3, … up to N.

Now let's say you have N data points $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$. Then the E in equation (**??**) is the sum of a bunch of terms, each of which looks like $(mx_i + b - y_i)^2$. That is difference between "what the fitted line would have given you" and "the real data point that you have". Squared. These differences are often called the "residuals".

So E is a pretty good measure of how poor your line fit is, so if you find the values of m and b that *make E as small as possible* then they might give you a nice straight line fit through your data.

To show you some more cool math typesetting, and to entice you to study calculus, I will show you the equations that are written to solve this problem:

$$\begin{gathered} E = \sum_{i=1}^{N} (mx_i + b - y_i)^2 \\ \frac{\partial E}{\partial m} = 0 \\ \frac{\partial E}{\partial b} = 0 \end{gathered} \tag{7.4.2}$$

This gives you two equations which you can solve to get m and b. We will not go through the details of it since it is calculus material ("finding minima using derivatives"), but we will now learn to use a Python library that does this calculation for us.

Note: There are many pleasing aspects to learning about least squares fitting. One of them is that this is one of the earliest places where you run in to … FIXME

7.5 Using Python's scientific libraries to fit lines

Python comes with an extensive scientific library called *scipy*. Scipy has a statistics subpackage, which in turn has a function called `linregress()` which does all that work for us.

Enter the program in

Listing 7.5.1: fit-height.py – fit a line through the height data and print out the slope and intercept.

```
#! /usr/bin/env python3

import sys
from scipy.stats import linregress

def main():
    if len(sys.argv) != 4:
        print('error: wrong number of arguments')
        print('usage: %s filename min_age max_age' % sys.argv[0])
        print('example: %s Howell1.csv 2 18' % sys.argv[0])
        sys.exit(1)
    fname = sys.argv[1]
    lowest_age = int(sys.argv[2])
    highest_age = int(sys.argv[3])

    xdata, ydata = load_file(fname, lowest_age, highest_age)
    slope, intercept, r_value, p_value, std_error = linregress(xdata, ydata)
    print('the least squares fit returns slope, intercept:')
    print('m =', slope, ', b =', intercept)


## load columns 2 and 0 from the file, return two data vectors.  only
## pick out ages betwen min_age and max_age (inclusive)
def load_file(fname, min_age, max_age):
    xdata = []
    ydata = []
    f = open(sys.argv[1], 'r')
    for line in f.readlines()[1:]:
        words = line.split(';')
        x, y = float(words[2]), float(words[0])
        if x >= min_age and x <= max_age:
            xdata.append(x)
            ydata.append(y)
    f.close()
```

(continues on next page)

(continued from previous page)

```
    return xdata, ydata

main()
```

Run the program with:

```
$ ./fit-height.py  Howell1.csv 2 18
the least squares fit returns slope, intercept:
m = 3.99026836995 , b = 79.6805438775
```

Now compare these values to those found in in Section **??** where we just picked two points to work with. It turns out that 3.99 is not too far from 4.12, and 79.68 is not far from 79.06, but they are different. Let us update Figure **??** to also show the line found by least squares linear regression:

Listing 7.5.2: Height vs. age, a naive line fit and a least squares fit for ages 2-18.

```
##REQUIRED_FILE: Howell1.csv
##PRE_EXEC: wget --continue https://raw.githubusercontent.com/rmcelreath/rethinking/
↪master/data/Howell1.csv
set datafile separator ";"
set grid
set key right bottom
set title 'height versus age for the !Kung people'
set xlabel 'age (years)'
set ylabel 'height (cm)'
plot [0:26] 'Howell1.csv' using 3:1 with points, \
    [2:18] 3.311 * x + 86.5867 lw 4 title "two point fit", \
    [2:18] 3.99026836995 * x + 79.6805438775 lw 4 title "least squares fit"
```

and you should get Figure **??**.

This line fit from the least squares method was a much better fit to the 2-18 year old data, so we can now say that the !Kung youth grow approximately 3.99 centimeters/year.

7.6 When to not try a linear fit

We should only try linear fits if we have reason to believe that the data is linear (either visually or from the scientific mechanism that underlies the data).

In the case of our height plot, what would happen if we were to try to fit the *entire* plot, from age 0 to the maximum age?

Let us try and see what it looks like. We run our program with ages 0 and 90:

```
$ ./fit-height.py  Howell1.csv 0 90
the least squares fit returns slope, intercept:
m = 0.909605221912 , b = 111.571782869
```

What would this line look like? Plotting like this:

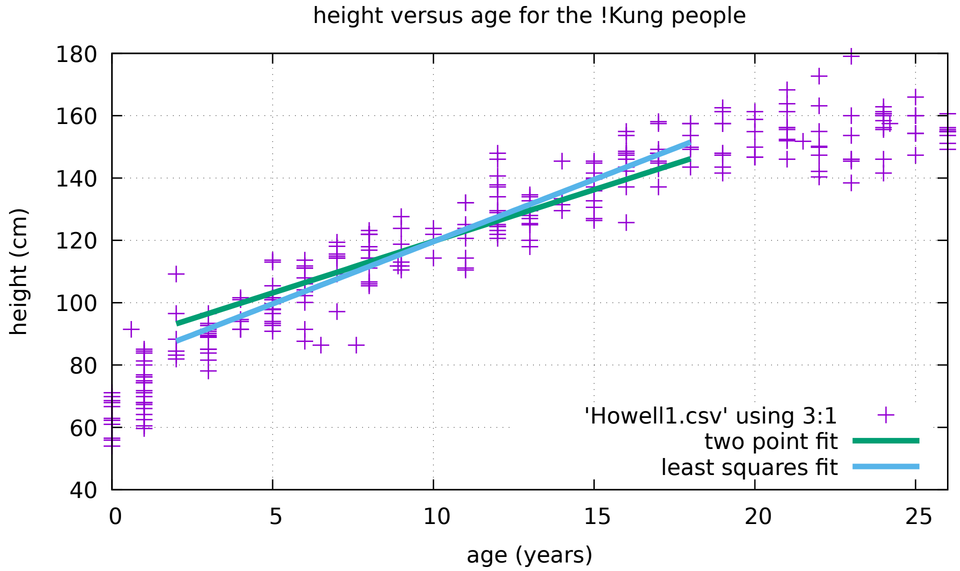


Figure 7.5.1: Height vs. age for the !Kung people of the Kalahari desert, with a naive line fit for ages 2-18, and a least squares fit as well. Note the difference between the two lines - the steeper line from the least squares fit is a better fit for the age 2-18 range.

Listing 7.6.1: Gnuplot instructions to plot an inappropriate line fit.

```
##REQUIRED_FILE: Howell1.csv
##PRE_EXEC: wget --continue https://raw.githubusercontent.com/rmcelreath/rethinking/
↪master/data/Howell1.csv
set datafile separator ";"
set grid
set key right bottom
set title 'height versus age for the !Kung people'
set xlabel 'age (years)'
set ylabel 'height (cm)'
plot 'Howell1.csv' using 3:1 with points, \
     0.909605221912 * x + 111.571782869 lw 4 title "least squares line fit"
```

and you should get the plot in Figure **??**.

We see that the mechanisms that determine height are quite different for infants, youths and older people, so there is no single line fit.

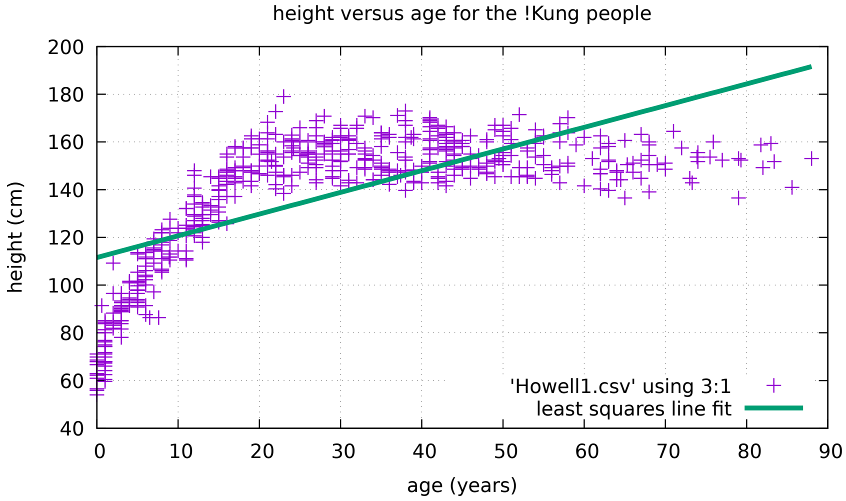


Figure 7.6.1: Height vs. age for the !Kung people of the Kalahari desert. We also plot a fit based on the entire plot, and we see that it is a poor fit to the data, since the data is not linear.

7.7 Fitting curves

7.7.1 Polynomial fits

Let us look at a falling body. In fact let us look at the first careful measurement that was ever made of a falling body. In the mid-17th-century Italian astronomer Riccioli [Gra12] carried out careful experiments from a tower in Bologna and got the data in this table:

Table 7.7.1: Riccioli's acceleration data

time	distance
0.833	10
1.66	40
2.50	90
3.33	160
4.17	250
1	15
2	60
3	135
4	240
4.33	280
1.08	18
2.17	72
3.25	162
4.33	280

You can download a CSV format file for this data: `riccioli-table.csv`

Newton's law with a constant acceleration of gravity $-g$ tells us that the position reached by a falling body from height h_0 (280 meters), with initial velocity $v_0 = 0$ is:

$$y(t) = h_0 + v_0 t - \frac{1}{2} g t^2 = h_0 - \frac{1}{2} g t^2 \tag{7.7.1}$$

When we calculate the coefficients for (**??**) we expect the first degree coefficient (initial velocity v_0) to be close to zero since we are letting it drop rather than tossing it. We also expect the constant term (initial height h_0) to be close to 280 roman feet, the height from which we drop it.

So we are interested in using the data from Riccioli's mid-17th-century experiment that we put in file `riccioli-table.csv`:

Listing 7.7.1: fit-falling-body.py - fit a 2^{nd} degree polynomial to the data, then plot the data and the fit.

```
#! /usr/bin/env python3

import sys
import numpy as np
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 10))

def main():
    if len(sys.argv) not in (2, 3):
        print(f'**error** usage: {sys.argv[0]} datafilename.csv [poly_deg]')
        sys.exit(1)
    input_fname = 'riccioli-table.csv'
    poly_degree = 2             # default is to fit a parabola
    if len(sys.argv) == 3:
        poly_degree = int(sys.argv[2])

    # use numpy's loadtxt() function to read the data columns
    t, y = np.loadtxt(input_fname, delimiter=',',
                      skiprows=1, unpack=True)
    # in our file the y values were listed as growing from 0, but we want
    # them to fall from the top, so we subtract them from 280 roman feet
    # get the polynomial fit
    y = 280 - y
    ## now do the polynomial fit
    print(f'fitting a {poly_degree} degree polynomial to the points')
    params = np.polyfit(t, y, poly_degree)
    print('polynomial parameters:', params)
    print('acceleration of gravity in roman feet/sec^2:', -2*params[0])
    ## note: in Riccioli's calculation one roman foot is probably 0.301
    ## meters
    print('acceleration of gravity in meters/sec^2:', -2*params[0]*0.301)
    plt.scatter(t, y, label='Data')
    ## now show the polynomial
    pfunc = np.poly1d(params)
    t_regular = np.arange(np.min(t), np.max(t+0.01), 0.01)
    y_from_poly = pfunc(t_regular)
    plt.title("Riccioli's falling body data")
    plt.plot(t_regular, y_from_poly, label=f'degree {poly_degree} polynomial fit')
    ## show the plots
    plt.xlabel('time')
    plt.ylabel('height (roman feet)')
    plt.legend(loc='best')
    ## you can call plt.show() to view it interactively, or plt.savefig()
    ## to save image files
```

(continues on next page)

(continued from previous page)

```
    #plt.show()
    assert(input_fname[-4:] == '.csv') # it should be a .csv file
    outfname_base = input_fname[:-4] + f'_deg{poly_degree}'
    save_plot_to_file(outfname_base, 'pdf')
    save_plot_to_file(outfname_base, 'svg')

def save_plot_to_file(outfname_base, fmt_suffix):
    out_fname = f'{outfname_base}.{fmt_suffix}'
    plt.savefig(out_fname)
    print(f'saved plot to file {out_fname}')

if __name__ == '__main__':
    main()
```

You can run the program with:

```
chmod +x fit-falling-body.py
./fit-falling-body.py riccioli-table.csv
```

It will find that the polynomial coefficients (a, b, c) are `-14.6918224, 0.62669318, 280.40412405` and the polynomial fit looks like:

$$y(t) = y = -14.69182224 \times t^2 - 0.62669318t + 280.40412405 \tag{7.7.2}$$

The program saves the output in files `riccioli-table_deg2.pdf` and `riccioli-table_deg2.svg`, and we show it in Figure **??**.

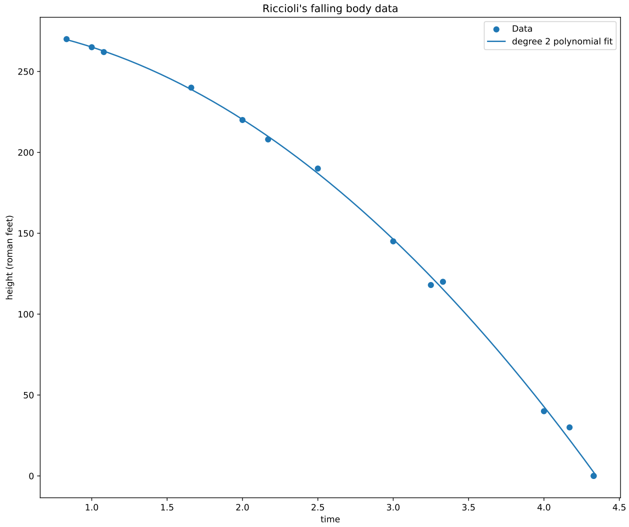


Figure 7.7.1: Position as a function of time for a falling body. The data is taken from Riccioli's classic experiment in the mid 17th century. The line is a 2^{nd} degree polynomial fit of the data.

The coefficients for the polynomial in equation (**??**) and Figure **??** teach us that:

- The 2^{nd} degree polynomial is a good fit to the data.

- The coefficients in the falling body equation (**??**) have physical meanings. The constant term h_0 is the height from which we drop the body, v_0 is the initial velocity, and in the $\frac{1}{2}gt^2$ term, g is the acceleration of gravity.
- In our fit we have a very small value for v_0 (close to 0), which matches the physical situation.
- In our fit we have h_0 very close to 280 roman feet, which matches the physical situation.
- In our fit we have g close to $29.38 \text{ roman feet}/s^2$. In modern units that gives us $g = 8.84\ m/s^2$, which is about 10% off of the current measured value of $9.81\ m/s^2$.

https://physicstoday.scitation.org/doi/pdf/10.1063/PT.3.1716

https://physicstoday.scitation.org/doi/full/10.1063/PT.3.1716

7.7.2 Overfitting

We all know that if you have two (distinct) points you can fit a *unique* straight line through them. If you write the equation for that straight line as $y = mx + b$ you might find yourself thinking "hmm, there were two points that constrained that line, and there are two parameters (m, b) (slope and intercept).

So two points gave unique values to two parameters. Is that "2" a coincidence?

It turns out that if you have three points in a plane, and they are not on a line together, then you can fit a *unique* parabola (2nd degree polynomial) through them. If that parabola looks like $y = ax^2 + b^x + c$, then you will notice that there are *three* parameters (a, b, c) being fixed by the *three* points that the curve must visit.

You might ask if you have 7 points, is there exactly one 6th degree polynomial that fits through all 7 points? The answer is yes, unless those 7 points are somewhat contrived. For example, if they all lie perfectly on a straight line, or on a parabola, or other lower order polynomial, then you will have *multiple* possible 6th degree polynomials that go through those pointes.

So you might ask the question:

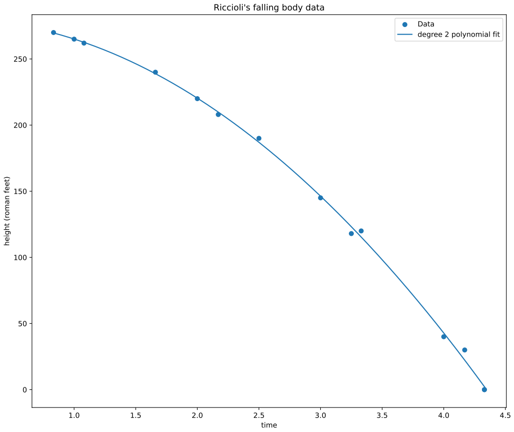
Note: Do I get better fidelity to the data if I fit a very high degree polynomial to the data? For example, if I take the Riccioli data set, and fit a 9th degree polynomial, will that capture some deep nuance in the data?

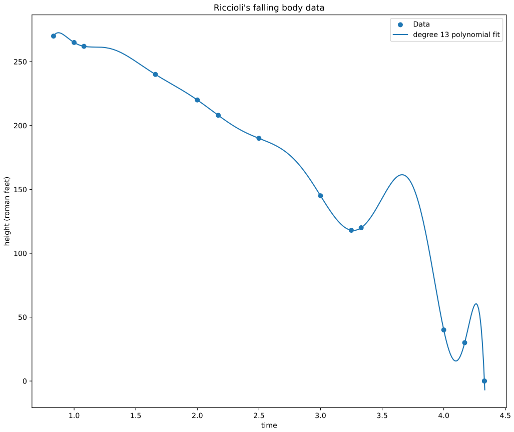
The answer is a clear *no*, and we will demonstrate that by trying it out and visualizing the results. After visualizing, we will try to gain some insight.

Start by modifying the program `fit-falling-body.py` to remove some of the data. It has 14 data points, so we could fit up to a 13th degree polynomial. We will do a few experiments by calling the program with arguments different from two. Start with:

```
./fit-falling-body.py riccioli-table.csv 13
```

This should give you a plot like the one shown here:





where you can clearly see that the 2nd degree polynomial makes more sense than the 13th degree polynomial.

So what just happened? The 13th degree polynomial is a *perfect* fit: it passes exactly through every point in the data set, so why is it so clearly bogus?

This is a good topic to discuss in class, and it leads to discussions of "overfitting".

7.7.3 Fitting arbitrary functions

Curve fitting is described here:

https://www.scipy-lectures.org/intro/scipy/auto_examples/plot_curve_fit.html

The equation we fit comes from Zhang, Jiang, Zhang, Liu, Wang and Loh "A Generalized SOC-OCV Model for Lithium-Ion Batteries and the SOC estimation for LNMCO Battery". [ZJZ+16]

$$V_{\mathrm{OCV}}(s) = a + b(-\log_e s)^m + cs + de^{n(s-1)}$$

where s is the "state of chage" (SOC) - how much charge the battery still has.

I found that with the data sets I had I needed to adapt it slightly and used:

$$V_{\mathrm{OCV}} = a + bs^{-m} + cs + de^{n(s-1)}$$

Each term in this equation corresponds to a different chemical effect in the battery's journey.

Listing 7.7.2: Battery discharge data: closed and open circuit voltage (volts) versus time (hours) for the discharge of two consumer-grade AA batteries in series. Source: Abby Wilson, private communication.

```
#time closed open
0     2.48   2.74
1     2.33   2.59
2     2.29   2.55
3     2.28   2.54
3.5   2.27   2.52
4.5   2.25   2.50
5.5   2.23   2.48
6     2.21   2.46
8     2.01   2.28
8.5   1.84   2.10
8.75  0.80   1.90
```

Listing 7.7.3: fit-battery.py – fit a function much like that in [ZJZ+16] to Abby Wilson's battery data.

```python
#! /usr/bin/env python3

import math
import sys
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

def main():
    if not len(sys.argv) in (2, 3):
        print('error: please give a battery data file argument')
        sys.exit(1)
    infile = sys.argv[1]
    outfile = None
    if len(sys.argv) == 3:
        outfile = sys.argv[2]

    times, OCVs = load_battery_file(infile)
    print('times:', times)
    print('open current voltages:', OCVs)
    p0=[3.5, -0.0334, -0.106, 0.7399, 1.403, 2]
    # p0=[2, 2, 2, 2, 2, 2]
    # SOCs = np.flip(times, 0) / (times[-1] - times[0])
    SOCs = times / (times[-1] - times[0])
    print('SOCs:', SOCs)
    params, params_covariance = optimize.curve_fit(Vocv, SOCs, OCVs, p0=p0)
    print('curve fit parameters:', params)
    print('a = %g' % params[0])
    print('b = %g' % params[1])
    print('c = %g' % params[2])
    print('d = %g' % params[3])
```

(continues on next page)

(continued from previous page)

```
    print('m = %g' % params[4])
    print('n = %g' % params[5])
    ## now visualize the results
    plt.figure(figsize=(6, 4))
    ## plot a scatter plot of the original data
    plt.scatter(SOCs, OCVs, label='Data')
    ## plot the function fit - use a denser range of times for this,
    ## since our time data is sparse
    # trange = np.arange(times[0], times[-1], 0.1)
    # SOCrange = np.arange(SOCs[0], SOCs[-1], 0.1)
    SOCrange = np.arange(0.0, 1.01, 0.005)
    plt.plot(SOCrange, Vocv(SOCrange, params[0], params[1], params[2],
                          params[3], params[4], params[5]),
             label='Fitted function')
    plt.xlabel('1 - state of charge (1 - SOC)')
    plt.ylabel('open circuit voltage (Vocv)')
    plt.legend(loc='best')
    if outfile:
        plt.savefig(outfile)
    else:
        plt.show()

def Vocv(SOC, a, b, c, d, m, n):
    # SOC = (t[-1] - t + 1) / (t[-1] - t[0])
    # print(a, b, c, d, m, n)
    # return a + b*(-np.log(SOC))**m + c*SOC + d*np.exp(n*(SOC-1))
    return a + b*SOC**(-m) + c*SOC + d*np.exp(n*(SOC-1))

def load_battery_file(fname):
    times, OCVs = np.loadtxt(fname, usecols=(0, 2), unpack=True)
    return times, OCVs

main()
```

Run this program with:

```
./fit-battery.py battery-discharge.dat
```

to get interactive graphical output, or to get the plot in a file you can use something like:

```
./fit-battery.py battery-discharge.dat battery-discharge.pdf
```

The resulting plot should look like that in Figure **??**.

The parameters found were:

```
a = 2.53651
b = 0.0512756
c = -0.182952
d = -0.499311
m = 0.151882
n = 15.5123
```

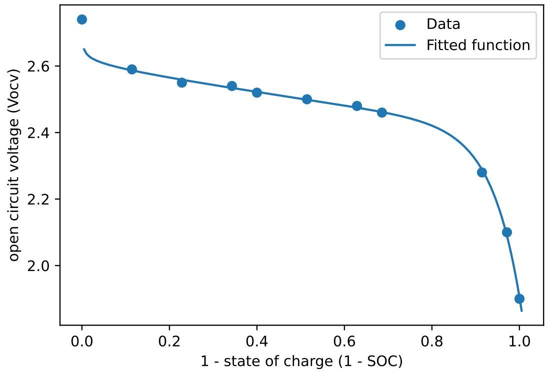


Figure 7.7.2: A curve fit for the battery discharge data.

So the fitted function looks like:

$$V_{\mathrm{OCV}}(s) = 2.54 + 0.051 s^{-0.152} - 0.183 s - 0.499 e^{15.5(s-1)}$$

There are many assumptions I made here. For example, I assumed that (1 - SOC) is linearly proportional to time as you discharge. That's not quite correct. But this is still an interesting example of how you can take a rather complicated function with various parameters (6 in this case) and fit it to a set of data.

7.8 Topics for further study

7.8.1 Interpolation and extrapolation

What do interpolation and extrapolation mean? Show an example where it works well, like with falling bodies.

Discuss what is a *predictive* model.

7.8.2 How high should the degree of the polynomial be?

Discuss overfitting and crazy polynomial behaviors.

Discuss "fitting unerlying truth" versus "fitting noise".

Discuss a physical reason for having a certain degree of polynomial (or a certain functional form), and what happens when you don't have that physical reason.

Discuss objective criteria for limiting the number of dimensions to avoid crazy polynomial behaviors.

Mention approaches that use functions other than polynomials, like Fourier analysis.

CHAPTER

EIGHT

CASE STUDIES IN DATA

8.1 Population data from the web

Our goals here are to:

- Automate fetching of data sets from the web.
- Look at a plot in a few different ways to get a narrative out of it.

We will start by looking at the population history of the whole world. When I discuss this with students I often ask "what do you think the population of the world is today?" (then you can have them search the web for "world population clock", which will take them to http://www.worldometers.info/world-population/).

Then ask "what do you think the world population was in 1914? And 1923? And 1776? And 1066? And in the early and late Roman empire? And in the Age of Pericles?

Let us search for

```
world population growth
```

and we will come to this web site: https://ourworldindata.org/world-population-growth/ and if we go down a bit further we will see a link to download the annual world population data. The text on the link is FIXME: this section is incomplete.

We will *not* click on the link. Instead we will use the program `wget` to download it automatically[5]:

```
$ wget http://ourworldindata.org/roser/graphs/[...]/....csv -O world-pop.csv
```

Note that this is a very long URL, but students can get it as a result of their search, so nobody has to type the full thing in.

Once they have the file downloaded they can look at the data with:

```
$ less world-pop.csv
```

and will quickly see that it is slightly different from the data we have seen so far. The columns of data are separated by *commas* instead of spaces. This type of file format is called *comma-separated-value* format and is quite common. Our plotting program, `gnuplot`, works with space-separated columns by default, so there are two tricks to plot the file. Either use the cool program `sed` to change the commas into spaces:

```
$ sed 's/,/   /g' world-pop.csv > world-pop.dat
$ gnuplot
gnuplot> plot 'world-pop.dat' using 1:3 with linespoints
```

[5] The full URL is http://ourworldindata.org/roser/graphs/WorldPopulationAnnual12000years_interpolated_HYDEandUN/WorldPopulationAnnual12000years_interpolated_HYDEandUN.csv but we don't need to type it all, so in the text I show an abbreviation of it.

or tell gnuplot to use a comma as a column separator:

Listing 8.1.1: Instructions to plot the world population.

```
##CAPTION: World population.
set grid
set datafile separator comma
plot 'world-pop.csv' using 1:2 with linespoints
```

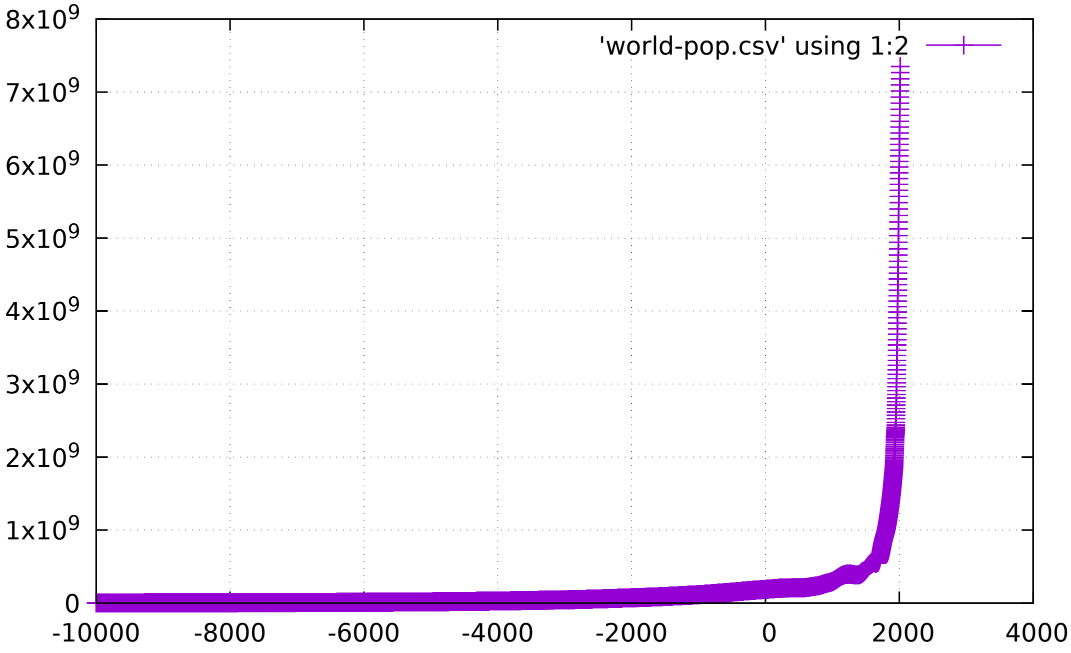


Figure 8.1.1: The world population from 10000 BCE until the present time.

And what a story we could tell from this plot if it weren't so hard to read! The main problem with this plot is that the world population in ancient times was quite small, and then it grew dramatically with various milestones in history which allowed for longer life expectancy and for the occupation of more of the world.

There are a couple of ways of trying to get more out of this plot. One is to *zoom in* to certain parts of it. For example, in we zoom in to the milennium from the founding of Rome to the fall of the western Roman empire, shown in Figure **??**.

Listing 8.1.2: Plot the world population from the founding of Rome until the fall of the western Roman empire.

```
##CAPTION: World population during the period of the Roman empire.
set grid
set datafile separator comma
plot [-753:476] 'world-pop.csv' using 1:2 with linespoints
```

This is a good time to stop and discuss the graph. In discussing Figure **??** students might make interesting connections referring to the Wikipedia Roman demography article It is sometimes estimated that the Roman empire might have had about 70 million citizens at the height of the empire, in the 2nd centry CE. The world population at that time was approximately 200 million people, so the Roman empire would have accounted for some 35% of the world's population. This means that large scale population events in the Roman empire, like the Antonine Plauge in 165-180 CE, or the decline and fall of the empire in the 4th and 5th centuries might account for dips in Figure **??**.

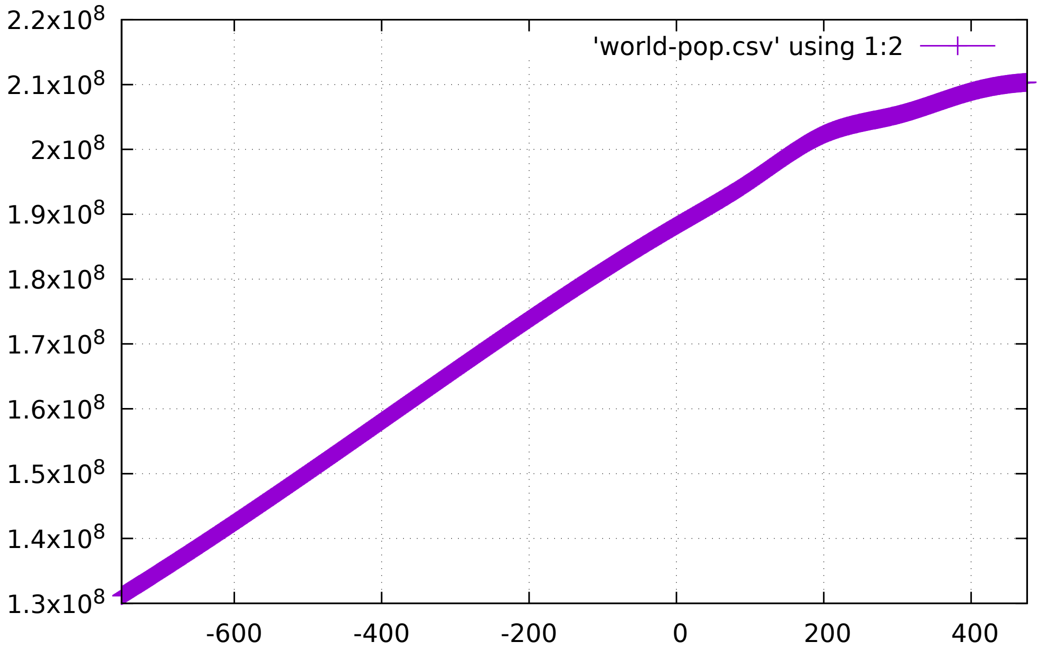


Figure 8.1.2: The world population from the founding of Rome (753 BCE) until the fall of the western Roman empire (476 CE).

We can also zoom in to the 20th century. In Figure **??** we zoom in to the 20th century.

Listing 8.1.3: Plot the world population in the 20th century.

```
##CAPTION: World population in the 20th century.
set grid
set datafile separator comma
plot [1900:1999] 'world-pop.csv' using 1:2 with linespoints
```

Discussion of Figure **??** can point out that there is exponential growth from 1900 to 1962 (the year in which the world's rate of population growth peaked), but that the exponential growth has interruptions due to World War I, the Spanish flu, and World War II.

Listing 8.1.4: Plot the world population from 0 to 1800 CE.

```
##CAPTION: World population from year 0 to 1800.
set grid
set datafile separator comma
plot [0:1800] 'world-pop.csv' using 1:2 with linespoints
```

And in Figure **??** we zoom in to the period from year 0 to 1800 CE. It can be interesting to look at pandemics and wars in this period and see if you can find features in the plot that correspond to those periods in history.

These attempts at zooming in tell us a some interesting things:

- It is frustrating that there is so little data before 1950.
- The 0 to 1800 plot allows us to see things clearly before the population jumps up so much.
- In the 0-1800 plot we see that the world population starts growing as we approach the year 1000, after which it flattens off around the year 1300 (the period of the great plague), after which it starts pick up and never stops

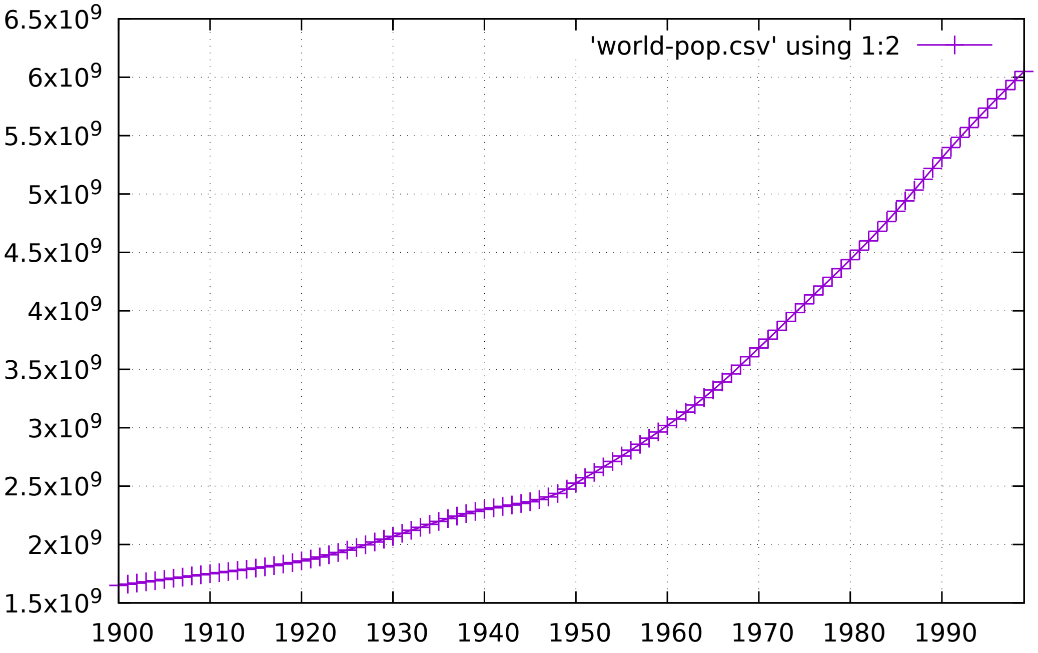


Figure 8.1.3: The world population in the 20th century.

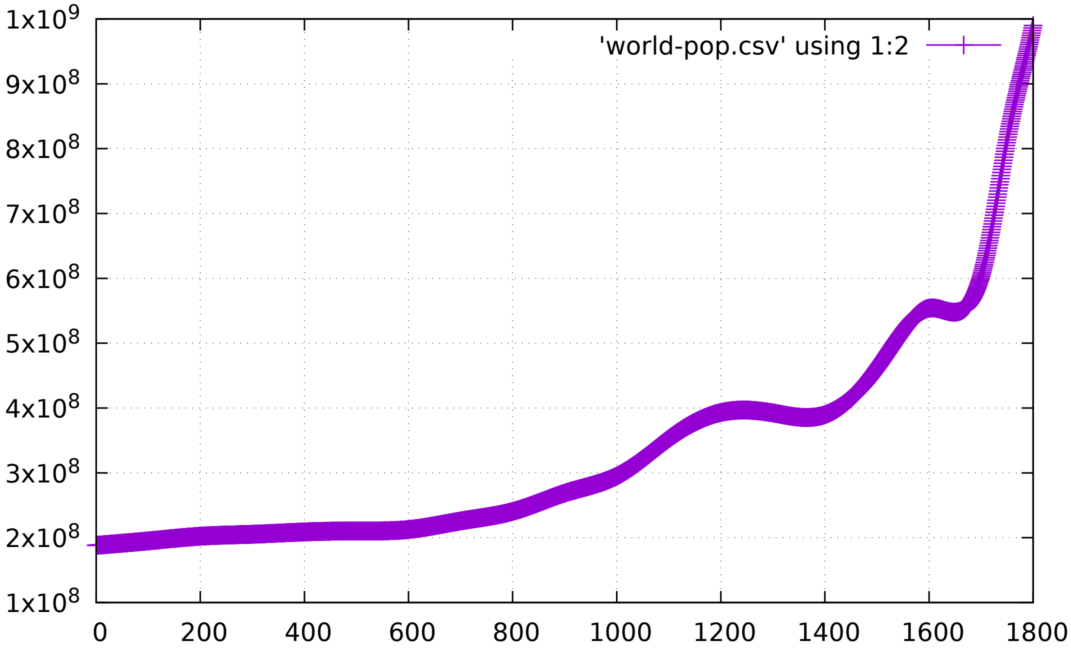


Figure 8.1.4: The world population from 0 to 1800 CE.

growing.

The other way to look at data when the y axis has too much range is to use what is called a *log scale*. Figure **??** shows how this can be done in `gnuplot`, and you can see that the y axis has been adjusted so that we can see some of the features in the data. This plot is more useful than that in Figure **??**.

Listing 8.1.5: Instructions to plot the world population with log scale.

```
##CAPTION: World population.
set grid
set datafile separator comma
set logscale y
plot 'world-pop.csv' using 1:2 with linespoints
```

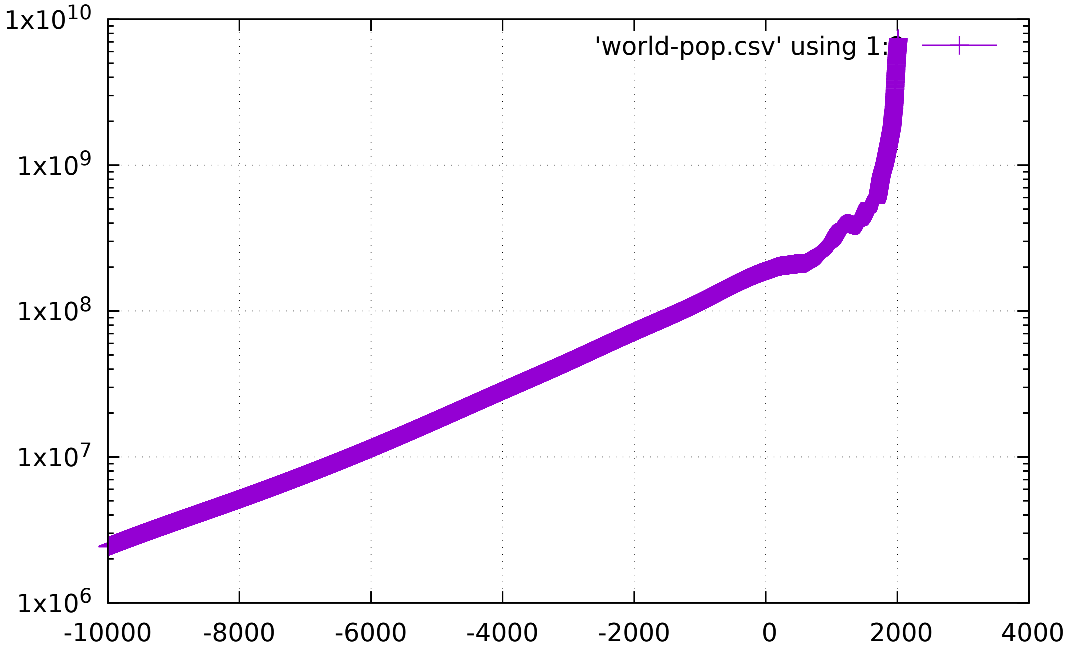


Figure 8.1.5: The world population from 10000 BCE until the present time, with a log scale for population. You can see some features because the log scale compresses the 20th century population explosion.

8.1.1 Exercises

Exercise 8.1 Find effective ways of downloading, processing and plotting data on the duration of ancient empires. You can find some here: http://www.bbc.com/future/story/20190218-the-lifespans-of-ancient-civilisations-compared

CHAPTER

NINE

SPECIAL NUMBERS: π

[status: written, but incomplete]

9.1 Motivation, prerequisites, plan

9.1.1 Motivation

Our purpose here is a recreational tour through the various aspects of the number π, including calculating it with the use of random numbers. This brings out various features of probability and statistics. This topic also gives us the opportunity to write several small programs that calculate π and do related things. It also allows us to go mention, in passing, various "higher math" aspects of π which get students comfortable with the terminology of those areas of math.

But this is not entirely recreational: the number π is deeply tied to the geometry of circles and spheres, and thus comes up in many physical laws: just think of *anything* that comes from a single point and spreads uniformly in all directions: the radiation from a (rather idealized) star, the electrical force field of an isolated electron, the gravitational field of an isolated mass. The geometrical aspect of these fluxes means that at a distance r from the origin you will see an intensity that is proportional to $\frac{1}{4\pi r^2}$

Among the physics formulae that involve π you will see:

- Einstein's equation for general relativity:

  $$G_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}$$

- The Heisenberg uncertainty principle for position and momentum in quantum mechanics:

  $$\Delta x \Delta p \geq \frac{h}{4\pi}$$

  Since π occurs so much in quantum mechanics, we have even coined a special constant and symbol called $\hbar$ (h-bar):

  $$\hbar = \frac{h}{2\pi}$$

  so that the uncertainty principle can be written as:

  $$\Delta x \Delta p \geq \frac{\hbar}{2}$$

9.1.2 Prerequisites

- The 10-hour “serious programming” course.
- The “Data files and first plots” mini-course in Section **??**
- Random number basics Section **??**

9.1.3 Plan

Our plan here is not a deeply rooted one: this is a playful romp through various aspects of π, with the intention of following whatever tangents might come up.

9.2 A collection of factoids

https://www.livescience.com/34132-what-makes-pi-special.html

among a collection of random whole numbers, the probability that any two numbers have no common factor — that they are “relatively prime” — is equal to $6/\pi^2$. Strange, no?

Finally, π emerges in the shapes of rivers. A river’s windiness is determined by its “meandering ratio,” or the ratio of the river’s actual length to the distance from its source to its mouth as the crow flies. Rivers that flow straight from source to mouth have small meandering ratios, while ones that lollygag along the way have high ones. Turns out, the average meandering ratio of rivers approaches — you guessed it — π.

https://www.angio.net/pi/whypi.html

https://www.newyorker.com/tech/annals-of-technology/pi-day-why-pi-matters

π is an *irrational* number: it can never be written as a fraction. In fact it is even more elusive: it can also never be written as the solution to a polynomial equation, so we say that it is not just irrational: it is also *transcendental*.

But there are a lot of cute *approximations* to π with fractions. The two which you might remember are:

$$\pi \approx \frac{22}{7} \approx 3.14285714286$$

and

$$\pi \approx \frac{355}{113} \approx 3.14159292035$$

For those who know some trigonometry, Machin’s formula is exact:

$$\frac{\pi}{4} = 4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

Another favorite of mine is *Stirling’s approximation* which gives an approximation to the factorial function (see also Section **??**):

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Stirling’s formula works for large-ish values of n, and approximates the exact value of $n!$ as $n \to \infty$. If you try it you will find that $10! = 3628800$, while Stirling’s formula gives 3598695.61893, which is off by about 0.8%.

9.3 Calculating π: ancient history

(FIXME must improve sources on this.)

Bible: fountain which is 30 paces around and 10 paces across.

Archimedes: inscribe polygons; go up to 6; he went up to 96. Depending on the energy level of the classroom, try to do the actual calculation!

9.4 Calculating π: monte carlo method

This is a good first introduction to *monte carlo* integration, which allows us to discuss monte carlo methods in general.

Introduce the European city of gambling, refer to James Bond, and then dive in to the method.

The method involves shooting darts into a square which has a circle inscribed in it. Draw the picture of a circle inside a square, and draw points of random darts hitting it.

The fraction of darts that fall in the circle is proportional to the fraction of areas:

$$\frac{N_{\rm cir}}{N_{\rm sq}} \approx \frac{A_{\rm cir}}{A_{\rm sq}}$$

The area of the circle is πr^2, and that of the square is πl^2. Since we have constructed this so that $l = 2r$ we get:

$$\frac{N_{\rm cir}}{N_{\rm sq}} \approx \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

This gives us:

$$\pi = 4\frac{N_{\rm cir}}{N_{\rm sq}}$$

Now what I usually do is write a live program which has a loop that throws 1000 darts. It does so by calculating `x = random.random() * 2 - 1` and the same for y. This gives us a dart in a square. Then using the pythagoras theorm with `if sqrt(x*x + y*y) < 1` we can determine if the dart is in the circle. We add all that up and estimate π.

I usually write this program (12 lines) live while the students write it with me. I write the program so that it prints, for each dart, four things: the index of the loop, the x coordinate, the y coordinate, and the estimate of π so far.

After experimenting with 1000 darts, then 100000, then a million, we go back to 1000 and redirect the output into a file.

This file can be plotted with a line using columns 1 and 4 (estimate of π vs. n_darts), and with points using columns 2 and 3 (the locations of the darts).

9.5 Calculating π: series that converge to π

Further reading: https://en.wikipedia.org/wiki/List_of_formulae_involving_%CF%80

Remember the terminology: a *sequence* $\{x_i\}$ is an ordered list of numbers with a criterion that gives you the next one in the sequence. An *infinite series* $\sum_{k=0}^{\infty} x_k$ is the sum of the sequence $\{x_k\}$. (Although you can then also say that the sequence of *partial sums* (i.e. up to N instead of up to infinity) of a *series* is a *sequence*, so the terms interweave…)

Over the years people have discovered many infinite series that converge to π.

9.5.1 Madhava-Leibniz series

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}$$

Properties: this series is exact, but it converges very slowly (*sublinear* convergence). To get 10 digits you need five billion terms of the series. See https://en.wikipedia.org/wiki/Leibniz_formula_for_%CF%80

9.5.2 "Efficient" infinite series

$$\frac{\pi}{2} = \sum_{k=0}^{\infty} \frac{2^k k!^2}{(2k+1)!}$$

The Bailey-Borwein-Plouffe formula has the interesting feature that it can be used to pick out any binary digit of π (although the digit extraction algorithm is lengthy):

$$\sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right)$$

Let us write a program which calculates this series up to a certain value of N:

Listing 9.5.1: series-to-N.py - sum a series up to N.

```
#! /usr/bin/env python3

import sys
import math

def main():
    """Calculate a series up to a certain value of k which is given on the
    command line
    """
    # series_term = series_term_leibniz
    series_term = series_term_zeta_2
    verbose = True
    try:
        lower = int(sys.argv[1])
        upper = int(sys.argv[2])
    except:
        print('usage: %s lower_bound upper_bound' % sys.argv[0])
        sys.exit(1)

    sum_to_here = 0
    for k in range(lower, upper+1):
        sum_to_here += series_term(k)
        # pi_approximation = 4 * sum_to_here
        pi_approximation = math.sqrt(6 * sum_to_here)
        if verbose:             # print the intermediate values
            print(k, '  ', sum_to_here, '  ', pi_approximation,
                  '  ', math.fabs(math.pi - pi_approximation), '  ',
                  math.fabs(math.pi - pi_approximation) / math.pi)

    print('result after %d iterations: %g' % (upper, sum_to_here))
```

(continues on next page)

(continued from previous page)

```
    ## FIXME: the formula below changes for different series
    print('## pi_approximation_final:', pi_approximation)


def series_term_leibniz(k):
    """Returns the kth term of the leibniz series for pi.  Sum starts at
    1.  Note that to print the resulting calculation of pi you will
    need to print 4*sum_to_here.
    """
    return (-1)**(k+1) / (2*k - 1)


def series_term_zeta_2(k):
    """Returns the kth term of the riemann function zeta(2).  Sum starts
    at 1.  Note that to print the resulting calculation of pi you will
    need to print sqrt(6.0*sum_to_here)."""
    return 1.0/(k**2)

## you can add more series term functions here

if __name__ == '__main__':
    main()
```

Note that the results of these series often need to then be square-rooted, or squared, or divided by something, so make sure to modify the `main()` function slightly each time so that you can see π clearly in the results. By default I have set it to multiply the summation by 4 for the Leibniz formula.

Try running this program and see how rapidly it converges to π. The Leibniz formula seems to require a factor of 10 more iterations to get just one more digit, which is quite slow.

Exercise 9.1 Right now `series-to-N.py` has (at least) two displeasing qualities: it asks you for the lower limit of the sum, and it requires that you change the code to print the resulting value of π depending on which series you are summing. Modify the program so that both those pieces of information are associated with the function, instead of having to give them as input or changing the code.

Exercise 9.2 For each of the infinite series you have programmed, make a plot of the convergence to π versus how many terms you sum. Then research the theoretical formulae that tell you how rapidly these series converge.

9.5.3 Formulae based on the Riemann zeta function

In general the Riemann zeta function is:

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

Evaluated at $s = 2$ we get the series:

$$\zeta(2) = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \cdots = \frac{\pi^2}{6}$$

and at $s = 4$ we get:

$$\zeta(4) = \frac{1}{1^4} + \frac{1}{2^4} + \frac{1}{3^4} + \cdots = \frac{\pi^4}{90}$$

Exercise 9.3 Write a function that calculates the Riemann zeta function for complex values of s and reproduce the attractive domain coloring plots shown in the images in https://en.wikipedia.org/wiki/Riemann_zeta_function

You will need to research how to calculate the zeta function for complex values.

9.6 Relationships between special numbers

One of the most striking math identities is *Euler's identity*:

$$e^{i\pi} + 1 = 0$$

It seems to combine the five most memorable numbers (0, 1, i, e, π) with the three basic arithmetic operations (addition, multiplication, exponentiation).

To see how mathematicians and philosophers have gone poetic on Euler's identity, see the wikipedia page: https://en.wikipedia.org/wiki/Euler%27s_identity#Mathematical_beauty

In class we can introduce a brief discussion of the Taylor series and show how the formula comes about, possibly adding the trigonometric identity to it. This will usually depend on a reading of the math level (and fatigue) of the students.

CHAPTER

TEN

A WORKSHOP ON PROGRAMMING BY YOURSELF (!)

Motivation

Here’s how I announce this mini-course to the mailing list:

```
This week on [...] I will teach a mini-course in which we practice
writing programs ourselves.  Most of our mini-courses involve a
program which I or one of the other students have written, but it's
important to break that umbilical cord and actually carry out some
exercises or write some programs from scratch.  So this week we will
do that: I will propose a program to write, but you can also pick
any program to work on, or exercise from the book.  You will work on
it on your own, but I will occasionally give tips, either to
individuals or to the whole class. [...]
```

Plan

I start out by proposing a problem. The one I have used gives some exposure to chemistry and an opportunity to explain some ideas.

CHAPTER

ELEVEN

RANDOM NUMBER BASICS

[status: unwritten]

Purpose:

- Learning more about random numbers

11.1 Prerequisites

- The 10 hour “serious programming” course.
- A GNU/Linux system with python3 and python3-tk installed (on an Ubuntu 16.04 system this can be done with `sudo apt install python3-tk`)

11.2 Motivation

In our “serious programming” course we used random numbers to program a random strategy in the computer’s tic-tac-toe algorithm.

It turns out that random numbers appear in very many areas of scientific computing, so in this chapter we will become comfortable with how to generate, plot and use random numbers.

11.3 Types of distributions

Here we will go a bit further and look at the various calls in Python’s `random` library, explore some simple . . . FIXME: to be filled

11.4 Further reading

[unfinished]

- https://www.youtube.com/watch?v=_tN2ev3hO14

CHAPTER

TWELVE

RANDOMNESS AND DISORDER

[status: content-mostly-written]

Purpose: to drive home the notion of disorder (and order) and how that relates to the probabilities of various situations.

Prerequisites:

a. The basic Python course.

b. Familiarity with simple plotting.

12.1 Experiment: burn a match

NOTE: this experiment should be carried out under adult supervision.

1. Have a flat piece of metal, or a tile, or a very flat rock. Lay it down in a stable place.
2. Light a match and *before* the flame reaches your finger, lay it gently on the flat metal or tile or rock.
3. Watch it until it finishes burning and let it cool down.
4. If possible, take the dark stick that is left and remake it into the original match.

12.2 Experiment: ink in water

1. Find an ink-like substance.
2. Fill a discardable plastic cup with water.
3. Drop a single drop of the ink into the water.
4. Observe the ink in the water the instant it falls in.
5. Observe the ink in the water after thirty seconds.
6. If possible, make the water return the drop of ink to where it was the instant it fell in.

12.3 Discussion on “ink in water” experiment

Discuss the meaning of the “ink in water” experiment with your partners. In particular discuss the meaning of the last step and whether it was possible.

12.4 Flipping a single coin

Take a coin and flip it 16 times. Tabulate the results like this:

```
H T T H H H T H T T H H T H T H ...
```

and so forth. Count how many times you get heads and how many times you get tails.

If you use 1 for heads and 0 for tails, calculate the average of all the tosses. In the few flips shown above it will be 0.5625: just a bit more than half of the tosses were heads.

12.5 Review: random numbers in Pythyon

To review random numbers, open the python interpreter with

```
$ python3
>>> import random
>>> random.random()
## repeat that many times
>>> random.randint(-3, 12)
>>> random.randint(-3, 12)
## repeat that many times
>>> random.randint(0, 1)
## repeat that many times
```

12.6 Experiment: flipping a single virtual coin

12.6.1 Just the flips

Flipping coins just a few times can give inconsistent results. Let us explore how long it takes for the average number of heads and tails to become consistent.

In this experiment we will write a computer program to flip a single virtual coin and look at the results. Then we will update the program to keep track of the average between heads and tails. We will count heads as 1, tails as 0, and print the average as we keep flipping. First enter (or paste) the program in Listing **??**.

Listing 12.6.1: single_coin_average.py

```python
#! /usr/bin/env python3
import random

def main():
    n_runs = 16
    n_heads = 0
```

(continues on next page)

(continued from previous page)

```
    n_tails = 0
    for i in range(n_runs):
        this_flip = random.randint(0, 1)
        if this_flip == 0:
            n_tails += 1
        else:
            n_heads += 1
        average = float(n_heads)/(n_heads + n_tails)
        print('%d    %f' % (i, average))

main()
```

Run this program with:

```
$ python3 single_coin_average.py
[... there will be output here ...]
$ python3 single_coin_average.py > coin_avg.dat
```

Then plot the results with:

```
$ gnuplot
# and at the gnuplot> prompt:
plot 'coin_avg.dat' using 1:2 with linespoints
plot [] [0:1] 'coin_avg.dat' using 1:2 with linespoints
```

Then change n_runs to be 100 and re-run the program and re-plot the results. Then plot 1000 runs.

12.6.2 Long-running average of single coin flips

Write the program in Listing **??** in a file called `single_coin_flip.py`

Listing 12.6.2: single_coin_flip.py

```
#! /usr/bin/env python3
import random

def main():
    n_runs = 16
    n_heads = 0
    n_tails = 0
    for i in range(n_runs):
        this_flip = random.randint(0, 1)
        print('%d    %d' % (i, this_flip))

main()
```

then run it with with:

```
$ python3 single_coin_flip.py
[... there will be output here ...]
$ python3 single_coin_average.py > coin_avg.dat
```

Then plot the results with:

```
$ gnuplot
# and at the gnuplot> prompt:
plot 'coin_avg.dat' using 1:2 with linespoints pt 7 ps 1
```

Then change n_runs to be 100 and re-run the program and re-plot the results. Then plot 1000 runs.

12.7 Flipping multiple coins

Take two different coins and flip them 16 times. Tabulate the results like this:

```
H T
T H
T T
H H
...
```

and so forth. Count how many times you get all heads and how many times you get all tails.

Do the same thing with three coins.

12.8 Experiment: flipping virtual coins

Since the purpose of computers is to *automate repetitive tasks*, we should not go beyond flipping three coins. Rather, we will write a computer program to do so. Enter the program in Listing **??** as `many_coins.py`:

Listing 12.8.1: many_coins.py

```
#! /usr/bin/env python3

"""
A simple program to simulate the flipping of several coins.  You
can experiment by chaning the variables n_runs and n_coins below.
"""

import random

def main():
    n_runs = 16
    n_coins = 1

    n_heads = 0
    n_tails = 0
    n_all_heads = 0
    n_all_tails = 0
    # outer loop is on how many runs we are doing
    for i in range(n_runs):
        this_flip = [0] * n_coins # store the results of a single run
        # now flip a bunch of coins for this run
        for coin in range(n_coins):
            flip = random.randint(0, 1)
```

(continues on next page)

(continued from previous page)

```
            this_flip[coin] = flip
            if flip == 0:
                n_heads = n_heads + 1
            else:
                n_tails = n_tails + 1
            print('%s ' % ('H' if flip == 1 else 'T'), end="")
        if this_flip.count(0) == n_coins:
            n_all_tails = n_all_tails + 1
        if this_flip.count(1) == n_coins:
            n_all_heads = n_all_heads + 1
        print('')
    print('after %d flips of %d coins, we got the following:'
          % (n_runs, n_coins))
    print('HEADS: %d' % n_heads)
    print('TAILS: %d' % n_tails)
    print('TOTAL: %d' % (n_heads + n_tails))
    print('RUNS_WITH_ALL_HEADS: %d' % n_all_heads)
    print('RUNS_WITH_ALL_TAILS: %d' % n_all_tails)
    print('fraction_runs_all_heads: %f' % (float(n_all_heads)/n_runs))
    print('fraction_runs_all_tails: %f' % (float(n_all_tails)/n_runs))


main()
```

The variables at the top, `n_runs` and `n_coins`, set how many runs you do and how many coins you flip in each run.

Experiment with `n_coins` = 2 and try to let `n_runs` go through 16, 50, and then try 1000. Run the program several times with each value of `n_runs` and pay close attention to the output lines `fraction_all_heads` and `fraction_all_tails` – are they more consistent when `n_runs` is larger?

Do the same with `n_coins` set to 3, 4, 5, and then to 20, keeping `n_runs` at 1000.

Discuss what you see happen to `fraction_all_heads` and `fraction_all_tails`.

12.9 Experiment: back to physical coins - disorder

1. Take 10 coins, set them up to be all heads and near each other on the ground.
2. Take a spatula, slide it under the 10 coins, toss them high up in the air.
3. Watch the debris and count heads and tails.
4. Take the spatula again and use a single movement of the spatula to put all the 10 coins back into their original state (all near each other and all heads). If you cannot do this with a single movement of the spatula, give yourself 10 spatula moves.
5. Repeat this experiment with the 10 coins stacked up instead “near each other”.
6. Now do what you have to do to restore the 10 coins to the pile where they are all “heads up” using the spatula. I that fails, do so with your fingers.

12.10 The drunk fencer

Let us now start talking about random walks. I want to move toward discussing random walks in 2 dimensions, but it is easier to program a one-dimensional random lurching back and forth - the way a drunk fencer might move back and forth randomly rather than according to the needs of the bout.

Let us type in the program in Listing **??**:

Listing 12.10.1: walk-1d.py – simulates a drunken fencer

```
#! /usr/bin/env python3

import random
import math
import sys

def main():
    x = 0
    n_steps = 10
    if len(sys.argv) == 2:
        n_steps = int(sys.argv[1])
    take_walk(x, n_steps)

def take_walk(x, n_steps):
    print(f'##COMMENT: sys.argv[0] - one dimensional random walk program')
    print(f'##TOTAL_STEPS_REQUESTED: {n_steps}')
    print(f'##COLUMNS: step_number, pos_x, sqrt(step_number), delta_pos_sqrt')
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        delta_x_sqrt_i = math.fabs(math.fabs(x) - math.sqrt(i))
        if i == 0:
            delta_x_sqrt_i_normalized = 0
        else:
            delta_x_sqrt_i_normalized = delta_x_sqrt_i / math.sqrt(i)
        print(i, x, math.sqrt(i), delta_x_sqrt_i, delta_x_sqrt_i_normalized)

main()
```

and run it and plot the results like this:

```
$ python3 walk-1d.py
$ python3 walk-1d.py > walk-1d.dat
$ gnuplot
# and at the gnuplot> prompt:
set grid
set size square
plot 'walk-1d.dat' using 1:2 with lines
```

By changing the number of steps to 100, 1000 and 10000 you should see the plots in Figure **??**.

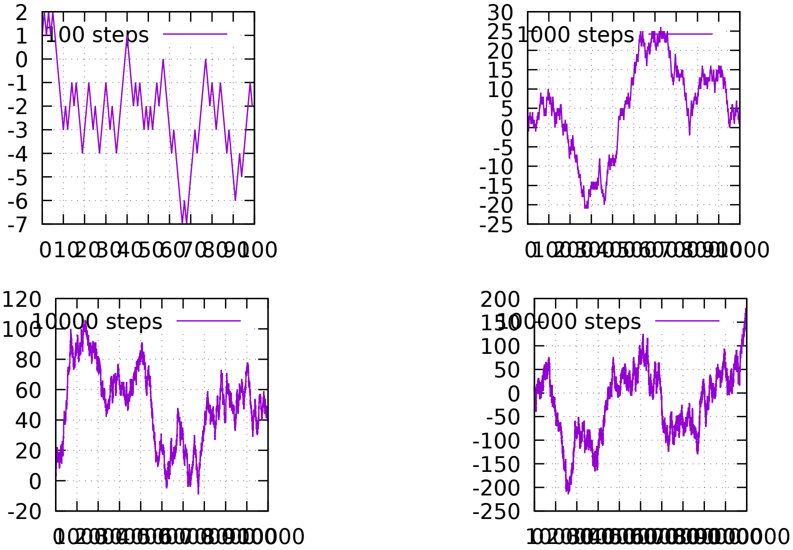


Figure 12.10.1: The path of a drunken fencer 100, 1000, 10000 and 100000 steps.

12.11 The drunkard’s walk

Now we move on to the more gratifying 2-dimensional random walk, also called the drunkard’s walk.

First introduce the notion with pictures and possibly an acting out of drunk behavior. Then type in the program in Listing **??**

Listing 12.11.1: walk.py

```
#! /usr/bin/env python3

import random
import math
import sys

def main():
    x = 0
    y = 0
    n_steps = 10
    if len(sys.argv) == 2:
        n_steps = int(sys.argv[1])
    take_walk(x, y, n_steps)

def take_walk(x, y, n_steps):
    print(f'##COMMENT: sys.argv[0] - two dimensional random walk program')
    print(f'##TOTAL_STEPS_REQUESTED: {n_steps}')
    print(f'##COLUMNS: step_number, pos_x, pos_y, distance_from_origin')
    for i in range(n_steps):
        ## generate steps of -1 or +1 in x and y directions
        step_x = 0
        step_y = 0
        ## use a coin toss to decide if we go in the x or y direction
```

(continues on next page)

(continued from previous page)

```
        if random.randint(0, 1) == 0:
            step_x = random.randint(0, 1) * 2 - 1
        else:
            step_y = random.randint(0, 1) * 2 - 1
        x = x + step_x
        y = y + step_y
        distance = math.sqrt(x*x + y*y) # distance from the origin
        print(i, x, y, distance)
        ## NOTE: if you want you can go farther and explore the way
        ## average distance scales with the number of steps (should be
        ## proprtional to the square root of the number of steps).
        ## for this it might be interesting to also print math.sqrt(i)

main()
```

Examples of running this program:

```
$ python3 walk.py
$ python3 walk.py 20
$ python3 walk.py 100
```

Now we want to do a big run and save the data to a file:

```
$ python3 walk.py 1000000 > walk-1000000.dat
```

Now we plot it. Note the tricks with the `tail` command which let you plot just the first few lines rather than the whole file:

```
$ gnuplot
# and at the gnuplot> prompt:
set grid
set size square
plot '<tail -100 walk-1000000.dat' using 2:3 with lines
plot '<tail -1000 walk-1000000.dat' using 2:3 with lines
plot '<tail -10000 walk-1000000.dat' using 2:3 with lines
plot '<tail -100000 walk-1000000.dat' using 2:3 with lines
plot '<tail -1000000 walk-1000000.dat' using 2:3 with lines
```

By changing the number of steps to 100, 1000 and 10000 you should see the plots in Figure **??**

12.12 Matplotlib animation of a random walk

Using matplotlib we can animate rather smoothly in real time. Try this program in Listing **??**:

Listing 12.12.1: walk_matplotlib.py

```
#! /usr/bin/env python3

"""This program plots an animated random walk using matplotlib.  You can run it with:
```

(continues on next page)

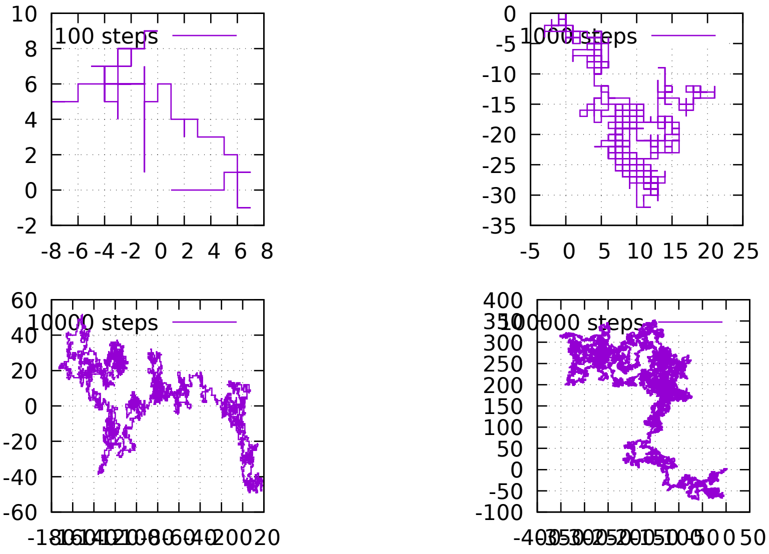


Figure 12.11.1: The path walked by a drunkard for 100, 1000, 10000 and 100000 steps. Note that for long walks the figure starts looking like a *fractal*.

(continued from previous page)

```

walk_matplotlib.py 500 1000

which would show 500 video frames to take 1000 steps.

A long run that shows how you can generate self-similar pictures could
be:

matplotlib.py 10000 100000

You could remove a zero from the first number to make it draw much faster.

"""

import sys
import math
import random

import matplotlib.pyplot as plt
import numpy as np

def main():
    n_frames = 500
    n_steps = 1000
    walk_file = 'walk_matplotlib.dat'
    if len(sys.argv) == 3:
        n_frames = int(sys.argv[1])
        n_steps = int(sys.argv[2])
    if n_frames > n_steps:
```

(continues on next page)

(continued from previous page)

```
        raise Exception('*error* cannot have more frames than steps')
    x = 0
    y = 0
    take_walk(x, y, n_frames, n_steps)
    plt.waitforbuttonpress()


def take_walk(x, y, n_frames, n_steps):
    """Takes a random walk, plotting the trajectory as we go."""
    assert(n_frames <= n_steps)
    draw_interval = int(n_steps / n_frames)
    assert(draw_interval > 0)
    print('## draw_interval: ', draw_interval)
    # some boiler plate stuff to prepare the matplotlib drawing area
    xmax = max(x, 10)
    ymax = max(y, 10)
    xmin = min(x, -10)
    ymin = min(y, -10)
    fig = plt.figure()
    ax = plt.axes()
    ax.set_xlim(-10, 10)
    ax.set_ylim(-10, 10)
    plt.grid(True)
    step_list = []
    full_path_x = []
    full_path_y = []
    line, = ax.plot(full_path_x, full_path_y, color='g', linewidth=3.0)
    # now that the plotting layout and variables are ready, we can
    # start the iteration
    for i in range(n_steps):
        step_x = 0
        step_y = 0
        prev_x = x
        prev_y = y
        ## generate steps of -1 or +1 in x and y directions
        ## use a coin toss to decide if we go in the x or y direction
        if random.randint(0, 1) == 0:
            step_x = random.randint(0, 1) * 2 - 1
        else:
            step_y = random.randint(0, 1) * 2 - 1
        x = x + step_x
        y = y + step_y
        # readjust the limits to account for where we are now
        xmax = max(xmax, x, ymax, y, -xmin)
        ymax = max(ymax, y, xmax, x, -ymin)
        xmin = min(xmin, x, ymin, y, -xmax)
        ymin = min(ymin, y, xmin, x, -ymax)
        ax.set_xlim(xmin, xmax)
        ax.set_ylim(ymin, ymax)
        distance = math.sqrt(x*x + y*y) # distance from the origin
        if i < 1:
            continue   # don't plot the first step
```

(continues on next page)

(continued from previous page)

```
        # our plotting approach will use the matplotlib
        # line.set_data() method, which uses a variable with all the
        # data in it.  this makes for smoother animation and better
        # memory use than other approaches, like drawing over previous
        # plots.
        full_path_x.append(x)
        full_path_y.append(y)
        prev_x = x
        prev_y = y
        # now do some clever balancing of how often we update the
        # drawing; this is based on the two input variables: the
        # number of frames and the number of steps.
        if i % draw_interval == 0:
            line.set_data(full_path_x, full_path_y)
            linewidth = 1 + 20.0 / (xmax - xmin)
            line.set_linewidth(linewidth)
            ax.figure.canvas.draw_idle()
            print(i, '   ', x, '   ', y, '   ', math.sqrt(x*x + y*y),
                  '   ', linewidth)
            plt.pause(0.000001)
    return step_list


main()
```

There are many things you might find notable in this program. The one I will comment on here is that there is a particular way in which we update the line in the plot (the key animation step).

Instead of using drawing commands to draw a new line, we use the `line.set_data()`. This is for speed: redrawing would be very slow, while updating the internal plot data will write to screen much faster.

Another thing to note is that in this example we don't use matplotlib's animation approach. We could probably change this program over, but it might be good to first see how to program it directly.

There is a tutorial on matplotlib animations here:

https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/

archived at:

https://web.archive.org/web/20220601192231/https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/

12.13 Making a movie from walk frames

12.13.1 Reviewing graphics and animation

In this session I will start with a discussion of graphical and video file formats: mention png (portable network graphics), jpeg (joint picture expert group), and mpeg (motion picture expert group).

We can use either `ffmpeg` or `convert` or `mencoder` to convert the sequence of frames into a movie.

In other mini courses, for example Section **??**, we have used the same idea of generating several individual frames and then using ffmpeg to make a movie out of them. We can go to that section and try those examples, and then return here.

It basically boils down to this. Find an image to work with, for example a Hubble Space Telescope image of the Pillars of Creation in the Eagle nebula:

```
wget https://ia902307.us.archive.org/7/items/pillars-of-creation-2014-hst-wfc-3-uvis-
↪full-res-denoised/Pillars_of_creation_2014_HST_WFC3-UVIS_full-res_denoised.jpg
# rename it something simpler
mv Pillars_of_creation_2014_HST_WFC3-UVIS_full-res_denoised.jpg pillars_of_creation_
↪hubble_big.jpg
# make it smaller
convert -resize 1000x pillars_of_creation_hubble_big.jpg pillars_of_creation_hubble.jpg
```

Then, to experiment with making animations, try this sequence:

```
for i in `seq -w 0 360`
do
    echo "rotating by $i degrees"
    convert pillars_of_creation_hubble.jpg -rotate $i pillars_of_creation_hubble-rotate$
↪{i}.jpg
done

ffmpeg -framerate 20 -i pillars_of_creation_hubble-rotate%03d.jpg rotate-movie-fr20.mp4
```

Then you can view `rotate-movie-fr20.mp4` with your favorite movie plyer.

12.13.2 Making individual frames of the random walk

We will use gnuplot to generate a sequence of images in png format.

Start with a file with a million lines of random walk output. You can do this as shown above with:

```
$ python3 walk.py 1000000 > walk-1000000.dat
```

gnuplot usually draws its output to the screen, but we can change that behavior and have it generate a `.png` file. To do so we add a couple of lines at the top:

```
$ gnuplot
# and at the gnuplot> prompt:
set grid
set size square
set terminal png
set output 'walk-100.png'
plot '<tail -100 walk-1000000.dat' using 2:3 with lines
set terminal png
set output 'walk-1000.png'
plot '<tail -1000 walk-1000000.dat' using 2:3 with lines
quit
# at the shell, possibly in another window, you can type:
$ ls
```

After running this last example you should find that there are a couple of new files in this directory: `walk-100.png` and `walk-1000.png`. If you view them with your favorite image viewer (you might want to consider the viewer `geeqie`) you will see that they are indeed the plots you wanted to make.

Our first task is to *automate* this process to generate hundreds or thousands of individual static images. The program `walk_movie.py` in Listing **??** does so.

Listing 12.13.1: walk_movie.py

```
#! /usr/bin/env python3

"""Makes a movie from a random walk output file."""

import os
import sys


def main():
    n_frames = 500
    n_steps = 1000000
    walk_file = 'walk-%d.dat' % n_steps
    if len(sys.argv) == 3:
        walk_file = sys.argv[1]
        n_frames = int(sys.argv[2])
        n_steps = int(os.popen("wc %s | awk '{print $1}'" % walk_file).read())
    print('Using %d frames from input file %s which has %d steps'
          % (n_frames, walk_file, n_steps))

    if not os.path.exists(walk_file):
        print('error: you must prepare the file %s -- you can do this with:'
              % walk_file)
        print('    ./walk.py %d > %s' % (n_steps, walk_file))
        sys.exit(1)

    gp_fname = 'walk_step.gp'
    for frame in range(0, n_frames):
        this_step = int(frame*n_steps/n_frames) # go in jumps
        this_step = max(this_step, 10)           # avoid small numbers
        png_fname = 'walk_frame-%06d.png' % frame
        if os.path.exists(png_fname):
            sys.stdout.write('## no need to rebuild %s        \r' % png_fname)
            sys.stdout.flush()
            continue
        gp = """set grid
set size square
set terminal png
set output '%s'
plot '<head -%d %s' using 2:3 with lines
quit\n""" % (png_fname, this_step, walk_file)
        f = open(gp_fname, 'w')
        f.write(gp)
        f.close()
        os.system('gnuplot %s' % gp_fname)
        percent = 100.0*frame / n_frames
        if frame % 10 == 0:
            sys.stdout.write('Making individual frames -- %.02f%% completed          \r'
↪% percent)
            sys.stdout.flush()
    print()
    print('## now you can make a movie with something like:')
```

(continues on next page)

(continued from previous page)

```
    print('ffmpeg -framerate 24 -i walk_frame-%06d.png walk-movie.mp4')
    print('## or:')
    print('ffmpeg -framerate 24 -i walk_frame-%06d.png walk-movie.ogg')
    print('## or:')
    print('convert walk_frame*.png walk-movie.mp4')
    print('## or for a more powerful encoder:')
    print('mencoder "mf://walk_frame*.png" -o walk-movie.mp4 -ovc lavc')
    print('## or in reverse:')
    print('ls -1 -r walk_frame*.png > reverse_filelist')
    print('mencoder "mf://@reverse_filelist" -o walk-movie-reverse.mp4 -ovc lavc')

main()
```

If you run this program it will generate `n_frames` different frames (the default in the program is 1000). To make a more satisfying movie we should increase this, but let us start with 1000 to make the program run more quickly.

`walk_movie.py` will pick out the random walk steps jumping 100 every time (see the line that says `n_steps = i*100`). It then generates a sequence of gnuplot commands like the one we saw above to generate file names that look something like `walk_frame-000023.png`.

If we run the program and the list the directory:

```
$ python3 walk_movie.py
$ ls
```

we see that the program has generated a bunch of `walk_frame-*.png` files (initially 1000 of them). Your favorite graphic viewer might allow you to press the space bar and almost see an animation of them.

Now let us talk about making a movie. There are several programs which *encode* a sequence of static image files into an mpeg movie. I mention three such programs: `ffmpeg`, `convert` and `mencoder`. The `walk_movie.py` program gave us a tip on how to run those programs to encode all the frames into the movie file `walk-movie.mp4`:

```
$ ffmpeg -framerate 24 -i walk_frame-%06d.png walk-movie.mp4
```

or

```
$ convert walk_frame*.png walk-movie.mp4
```

or

```
$ mencoder "mf://walk_frame*.png" -o walk-movie.mp4 -ovc lavc
```

or, to make a movie in reverse:

```
$ ffmpeg -framerate 24 -start_number -999999 -i walk_frame-%06d.png walk-movie.mp4
## FIXME: the reverse order with a negative start number needs to
## be ironed out.  Maybe ffmpeg can take an `ls -r ...` on the
command line.
```

or

```
$ ls -1 -r walk_frame*.png > reverse_filelist
$ mencoder "mf://@reverse_filelist" -o walk-movie-reverse.mp4 -ovc lavc
```

Note that the simplest and fastest approach is to use `ffmpeg` (the “Swiss army knife” of video and audio formats) so I will continue with `ffmpeg` for my examples.

You can now play these movies with your favorite movie player - I recommend `vlc`, though your system might come with `totem` already installed:

```
$ vlc walk-movie.mp4
$ vlc walk-movie-reverse.mp4
```

I wrote `walk_movie.py` to only generate 500 frames so that it can run quickly when I give examples or when I build this book, but you should increase that number a lot so you can see a longer movie with more detail.

Playing the movie shows you the growth of a fractal. It is interesting to watch how sometimes it adds paths in a clump, while sometimes it darts off into another sector of the plane, creating some filaments that connect the thicker bushes.

12.14 Discussion

Discuss the result of these experiments with your partners. Ask each other the following questions and write slides to present your answers. You might write one slide for each few related questions. You may include any of the plots you made in this course into your slides.

- Look at the first two plots we made. In the first one (single coin flips) note that a new point in the plot is completely independent of the previous ones. In the second one (running average of single coin flips) is each new point on the plot related to previous ones? Are there two types of random events - those that are fresh and new, and those that add a small random change to a previous state? The former is called a *Poisson process*, the latter is called a *Markov process* or a *Markov chain*.
- Look at the plot of averages from the single coin experiment and tell a story of what is going on in that plot, addressing the fact that initially it fluctuates quite a bit and later it seems to converge to 0.5.
- Is this related to how many more ways you can create 10 disordered coins than 10 ordered coins?
- Was it more work to restore the 10 coins to "heads up" or was it more work to toss them with the spatula.
- Why it easier to restore 10 coins to their initial "heads up" state than to restore the drop of ink to its inital state?
- Is it easier to go from order to disorder or from disorder to order?
- Have you heard of *Entropy*? If not, now you have. Entropy is related to the idea of *disorder*. One of the deepest laws in physics is the *Second Law of Thermodynamics*. One of the ways of stating the second law is that *the entropy of the universe is always increasing*.
- Referring back to the *Emergent Behavior* short course, remember how in that situation we examined systems that go from chaos to order: a random first row in a cellular automaton becomes a pattern of triangles, and a random initial state in John Conway's game of life can produce ordered patterns as well as gliders. Discuss how emergent behavior seems to produce order out of chaos - does this violate the second law of thermodynamics?
- Discuss the cycle that brings to living creatures: trees, flowers, cats, dogs, horses, humans… Is each living creature more or less ordered than the molecules in the earth and air from which that creature is made?

12.15 Further reading and videos

A cartoon video for kids on entropy:

https://www.youtube.com/watch?v=Tay3-2WKQ5Y

Jeff Phillips's TED-ed video "What is entropy?":

https://www.youtube.com/watch?v=YM-uykVfq_E

The "What is Entropy?" video from "The Good Stuff":

https://www.youtube.com/watch?v=gS_C7dM25pc

A discussion of how information in the demon's brain makes it respect the second law is here:

https://www.youtube.com/watch?v=ULbHW5yiDwk

Science asylum video on entropy:

https://www.youtube.com/watch?v=ykUmibZHEZk

Try this:

https://www.youtube.com/watch?v=lj5tqM5GZnQ

starting at minute 14:10

Feynman's lecture on "The Distinction of Past and Future":

https://www.youtube.com/watch?v=_Kab9dkDZJY

CHAPTER

THIRTEEN

RANDOM PROCESSES

[status: partially-written]

13.1 Motivation, prerequisites, plan

Motivation

There is a surprising depth in understanding various aspects of random processes, also called *stochastic processes*. The wikipedia article mentions that:

> *"They have applications in many disciplines including sciences such as biology, chemistry, ecology, neuroscience, and physics as well as technology and engineering fields such as image processing, signal processing, information theory, computer science, cryptography and telecommunications. Furthermore, seemingly random changes in financial markets have motivated the extensive use of stochastic processes in finance."*

This is a staggering list of fields: random processes seem to be as pervasive in all fields of physical, biological and social science as basic math, statistics and calculus.

Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**.
- Random number basics from Section **??**.
- The sections on random walks from Section **??**.

Plan

So how do we write programs that study and visualize these ideas? We will:

1. Review random number generation.
2. Introduce simple poisson processes: random amplitude and random time between samples.
3. Examine the distribution of Δt in a poisson process.
4. Look at random placements on a 2D lattice.
5. Various types of time series.
6. Revisit random walks.

7. Diffusion: distribution of distance covered in a random walk after time t.
8. Brownian motion and kinetic theory.
9. Progression of record peaks, and progression of sports world records.

13.2 Reviewing random number generation

You should quickly review the brief section on what web pages look like in Section **??** before continuing in this section.

Now let us write a few lines of code which generate random numbers and then puts them into a file which we can analyze and plot.

```
$ python3
>>> import random
>>> random.random()
>>> random.random()
>>> [...]
>>> ## now put the numbers in a file
>>> f = open('time_series.out', 'w')
>>> f.write('## time  time_interval  what_happened\n')
>>> t = 0
>>> for i in range(300):
>>>     dt = random.random()
>>>     t = t + dt
>>>     f.write('%g  %f  EVENT!!\n' % (t, dt))
>>> f.close()
```

[not sure about the remaining paragraphs in this section; they are commented out for now]

13.3 Poisson processes

A poisson process is a random process in which each event's probability is *unrelated* to any previous events.

Examples include throwing dice, flipping coins, radioactive decay.

We will write programs that simulate various types of poisson process.

13.3.1 A pure poisson process

Our first program will

13.3.2 An angry lightning goddess

Scenario: you live in a very unfortunately placed house. *Every day* Astrape, the goddess of lightning, rolls her special dice and has a probability of 0.03 (3%) to hitting that house. This should correspond to an average of one hit per month.

Let us first write a program which generates this series of events using a random number generator. Our first stab at this program will simply see that the probability was indeed close to 0.03.

Listing 13.3.1: lightning-first-stab.py – Hit a house with lightning with a 3% chance every day. First stab at simulating this.

```python
#! /usr/bin/env python3

"""Simpole example of simulating lightning strikes with a fixed
probability every day.  This first stab at a simulation simply
calculates the probability of the hit to make sure that the random
number generator performs as we expect.

"""

import random

def main():
    n_days = 1000
    n_hits = 0
    n_misses = 0
    for day in range(n_days):
        r = random.random()
        if r <= 0.03:            ## 3% chance
            n_hits += 1
        else:
            n_misses += 1
    hit_fraction = n_hits / n_days
    print('average daily hits: %g (%d days)' % (hit_fraction, n_days))

if __name__ == '__main__':
    main()
```

Run the program a few times to see that you indeed got

The key mechanism used in this program was in these lines of code:

```python
    # ...
    r = random.random()
    if r <= 0.03:            ## 3% chance
        n_hits += 1
    else:
        n_misses += 1
    # ...
hit_fraction = n_hits / n_days
```

Make sure you understand that the `random.random()` call returning less than 0.03 happens 3% of the time.

Now, following an example from Steven Pinker [FIXME: put citation to The Better Angles of Our Nature], we ask the question:

> If Astrape struck your house today, which day is the most likely day for the *next* bolt of lightning to strike your house?

To estimate this we write the following program:

Listing 13.3.2: lightning-time-distribution.py – Hit a house with lightning with a 30% chance every day. Estimate the distribution of time intervals between strikes.

```
#! /usr/bin/env python3
import random
import math

def main():
    ## run this program with n_days = 50 when you want
    ## to eyeball the output; run it with n_days = 1000,
    ## then 10*1000, then 100*1000 when you want to make
    ## plots
    n_days = 100000
    delta_t_list = simulate_strikes(n_days)
    ## now that we have the list we print it to a file
    with open('time_diffs.dat', 'w') as f:
        for delta_t in delta_t_list:
            f.write("%d\n" % delta_t)
        print('wrote time_diffs.dat with %d delta_t values'
              % len(delta_t_list))


def simulate_strikes(n_days):
    """simulates lightning strikes for a given number of
    days, collecting information on the times between
    strikes. returns the list of delta_t values.
    """
    last_delta_t = -1
    delta_t_list = []
    prev_day_with_strike = -1
    for day in range(n_days):
        r = random.random()      # a random float between 0 and 1
        if r <= 0.03:            # 3% chance
            #print('%d: hit' % day)
            if prev_day_with_strike >= 0:
                ## we record the delta_t of this event
                last_delta_t = day - prev_day_with_strike
                delta_t_list.append(last_delta_t)
            prev_day_with_strike = day
    return delta_t_list

if __name__ == '__main__':
    main()
```

You can then plot the results of this program in a couple of different ways. We will first make a *scatter plot* showing the lightning events as points in a plot with lightning number and Δt. You can see this in Figure **??**.

Listing 13.3.3: Instructions to make a scatter plot of Δt for each pair of lightning strikes.

```
##REQUIRED_FILE: time_diffs.dat
##PRE_EXEC: ./lightning-time-distribution.py
```

(continues on next page)

(continued from previous page)

```
set xlabel 'lightning number'
set ylabel '{/Symbol D} t (days)'
plot 'time_diffs.dat' using 1 with points pt 4 \
    ps 0.3 title 'time between lightning strikes'
```

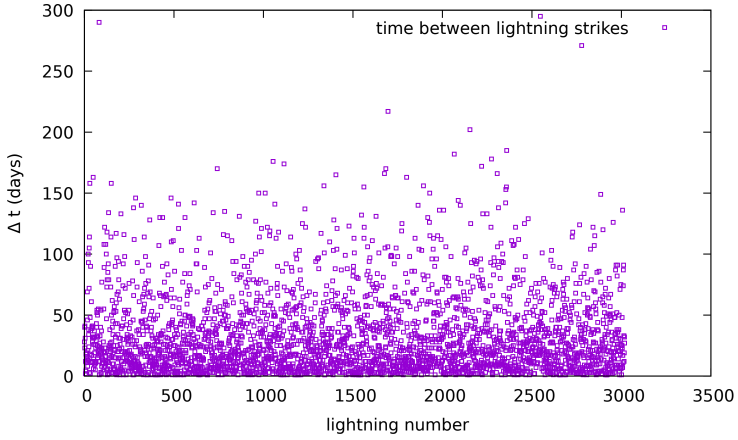


Figure 13.3.1: Scatter plot of Δt for each pair of lightning strikes. Notice that there are more strikes in the lower portions of the plot, which means that many strike pairs are close by!

A bit more rigor and insight can be gotten from a *histogram* plotting the number of lighting pairs versus the time between the two strikes. This histogram is shown in Figure **??**.

Listing 13.3.4: Instructions to make a histogram plot of lightning pairs versus Δt.

```
##REQUIRED_FILE: time_diffs.dat
##REQUIRED_FILE: time_diffs.dat.hist
##PRE_EXEC: ./lightning-time-distribution.py
##PRE_EXEC: ../plotting-intermediate/int-histogram-maker.py time_diffs.dat
set grid
set xlabel '{/Symbol Delta} t (days)'
set ylabel 'frequency of that interval'
set style data histogram
set style fill solid 0.8 border -1
plot [0:] [0:] 'time_diffs.dat.hist' using 1:2 with boxes
```

First conclusion: the answer to Steven Pinker's question is that the most likely day in which the *next* lightning will strike your house is *tomorrow*. The goddess Astrape likes to play with humans.

Another conclusion: the distribution of time between consecutive lightning strikes looks like a *decaying exponential*.

[…many other conclusions…]

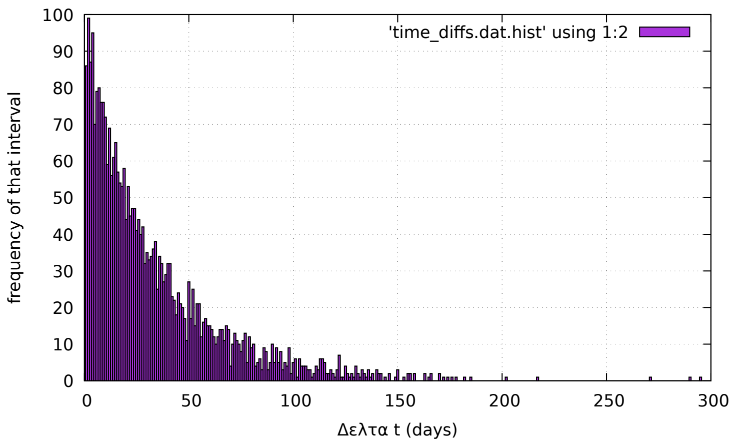


Figure 13.3.2: Histogram of number of lightning pairs versus Δt. This shows even more clearly that there are many more consecutive lightning pairs that are closely spaced.

Questions

Now muse about the time distance between droughts and other natural calamities.

Examples: "Climate Change and Cultural Response" by Benson and Berry, page 105, shows distribution of megadroughts in Figure 7.

Look in to the "palmer drought severity index" (PDSI) to find datasets. https://www.ncdc.noaa.gov/temp-and-precip/drought/historical-palmers/

PDSI is often shown from 1850 or 1870 until today. Some data sets go older, using tree ring data, and show the interesting periods from 800 to 1400, which would allow us to study the collapse of the Chaco Canyon culture.

For information about file formats and to see how pervasive obsolete data formats are, look at ftp://ftp.cdc.noaa.gov/Datasets/dai_pdsi/README

Download the .nc files at ftp://ftp.cdc.noaa.gov/Datasets/dai_pdsi/ and examine them with `ncview`.

Or maybe better: get the ascii files from here: http://www.cgd.ucar.edu/cas/catalog/climind/pdsi.html for example http://www.cgd.ucar.edu/cas/catalog/climind/pdsisc.monthly.maps.1850-2010.fawc=1.r2.5x2.5.ipe=2.txt.gz

For Kansas look at http://www.kgs.ku.edu/Hydro/Publications/2012/OFR12_18/KGS_OF_2012-18.pdf page 10, Figure 5a: PDSI for 800 to 2000. And Figure 5b for megadroughts. Figure 5c shows the dustbowl drought.

Aha: the paleoclimatology (2000 years) might be here: https://www.ncdc.noaa.gov/paleo-search/study/19119 and a text file here: https://www1.ncdc.noaa.gov/pub/data/paleo/drought/LBDA2010/LBDA_PMDI_JJA_Recons.txt and the grid metadata is explained here: https://www1.ncdc.noaa.gov/pub/data/paleo/drought/LBDA2010/LBDA_PMDI_Instrumental_metadata.txt Note that Chaco Canyon Lat/Lon are: 36.0530 deg N, 107.9559 deg W. From the meatadat it looks like it might be grid points 1883 or 1884 in this data set.

Q: Are drought cycles random? Are they the combination of so many types of physics that they appear to be random enough? How about anthropogenic climate change? Does it add a baseline?

Topic: Atlantic multidecadal oscillation.

13.3.3 Vicious glow-worms

We have talked about processes which give events distributed *randomly in time:* events happen at random times. Let us now look at processes that generate points distributed *randomly in space:* (x, y) coordinates are spewed out by our process. An example might be where the grains of sand land when you drop a handful onto the ground.

We can write a program to generate random (x, y) points between 0 and 100. The program `random-spatial.py` generates a series of such points, each completely independent of the previous one.

Listing 13.3.5: random-spatial.py – Generate a sequence of random (x, y) points.

```
#! /usr/bin/env python3

"""
Print a bunch of (x, y) points.
"""

import random

def main():
    for i in range(3000):
        x = random.random() * 100.0
        y = random.random() * 100.0
        print('%f    %f' % (x, y))

if __name__ == '__main__':
    main()
```

You can plot this with:

Listing 13.3.6: Instructions to plot points randomly distributed in a two-dimensional space.

```
##REQUIRED_FILE: random-spatial.dat
##PRE_EXEC: ./random-spatial.py > random-spatial.dat
##CAPTION: Random (x, y) points.  You should be able to see
##CAPTION: some structure: occasional filaments, clustering, and empty
##CAPTION: spaces.
set size ratio -1
set xlabel 'x'
set ylabel 'y'
plot 'random-spatial.dat' using 1:2 with points pt 7 \
    ps 0.3 title 'random (x, y) points'
```

The results are shown in Figure **??**. You can see *features* in the data, even though it was randomly generated: filaments, clustering, voids…[1]

A possible comment: people who spend a lot of time looking at randomly generated data probably don't easily believe in conspiracy theories.

We can then do something analogous to what we did for events with random time intervals: plot the distribution of distances between (x, y) points. The programs `xy-to-distances.py` and `float-histogram-maker` allow us to do so, and the results are in . Note that you will not get as much insight out of these spatial histograms as you did in , since a big factor in the distribution of spacial distances is the small size of the *x*-*y* plane we used.

[1] Note that the clustering is an artifact of the random generation of points; it is not due to a physical effect that clusters the points together.

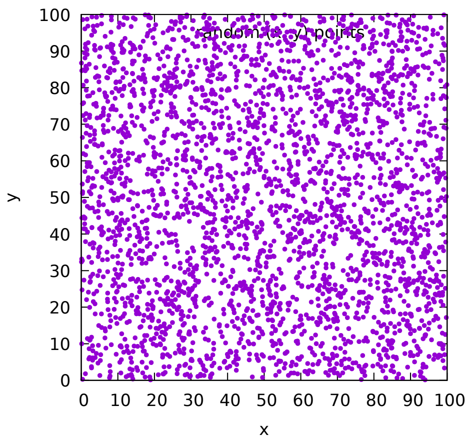


Figure 13.3.3: Points randomly distributed in a two-dimensional space.

Before running these programs you will need to install the `python3-scipy` package:

```
$ sudo apt install python3-scipy
```

Listing 13.3.7: xy-to-distances.py – Convert a collection of (x, y) points to a list of distances between points.

```python
#! /usr/bin/env python3

"""Load a collection of (x, y) points and print out the distance
between each pair of points"""

import sys
import scipy
import scipy.spatial
import math

def main():
    fname = sys.argv[1]
    coords = []
    with open(fname, 'r') as f:
        lines = f.readlines()
        for line in lines:
            xs, ys = line.split()
            (x, y) = (float(xs), float(ys))
            coords.append((x, y))
    print('read in %d coordinates' % len(coords))

    dist_out_fname = sys.argv[1] + '.distances'
    nearest_fname = sys.argv[1] + '.nearest'
    with open(dist_out_fname, 'w') as fpairs:
        with open(nearest_fname, 'w') as fnearest:
            for i in range(len(coords)-1):
                nearest_distance = sys.float_info.max
                for j in range(i+1, len(coords)):
```

(continues on next page)

(continued from previous page)

```
                    r = math.hypot(coords[j][0] - coords[i][0],
                                   coords[j][1] - coords[i][1])
                    ## write all pairwise distances to the fpairs file
                    fpairs.write('%g\n' % r)
                    ## now see if we have a new nearest distance
                    if r < nearest_distance:
                        nearest_distance = r
                ## write nearest distances to a separate file
                fnearest.write('%g\n' % nearest_distance)
    print('wrote distances to %s' % dist_out_fname)
    print('wrote nearest distances to %s' % nearest_fname)

if __name__ == '__main__':
    main()
```

You can run this with:

```
$ ./random-spatial.py > random-spatial.dat
$ ./xy-to-distances.py random-spatial.dat
```

and plot the result with:

Listing 13.3.8: Instructions to plot a histogram of distances between points randomly distributed in a two-dimensional space.

```
##REQUIRED_FILE: random-spatial.dat
##REQUIRED_FILE: random-spatial.dat.distances
##REQUIRED_FILE: random-spatial.dat.nearest
##REQUIRED_FILE: random-spatial.dat.distances.hist
##REQUIRED_FILE: random-spatial.dat.nearest.hist
##PRE_EXEC: ./random-spatial.py > random-spatial.dat
##PRE_EXEC: ./xy-to-distances.py random-spatial.dat
##PRE_EXEC: ./float-histogram-maker.py random-spatial.dat.distances
##PRE_EXEC: ./float-histogram-maker.py random-spatial.dat.nearest
set grid
set multi layout 2,1
set style data histogram
set style fill solid 0.8 border -1
set xlabel 'distance between random (x, y) points'
set ylabel 'frequency of that distance'
plot [0:] 'random-spatial.dat.distances.hist' using 1:2 with boxes
set xlabel 'nearest distance to a point'
set ylabel 'frequency of that nearest distance'
plot [0:] 'random-spatial.dat.nearest.hist' using 1:2 with boxes
```

The results are shown in Figure **??**.

So we have seen that in two spatial dimensions we have a situation analogous to that for time differences: nearest points are distributed with a *decaying exponential* distribution.

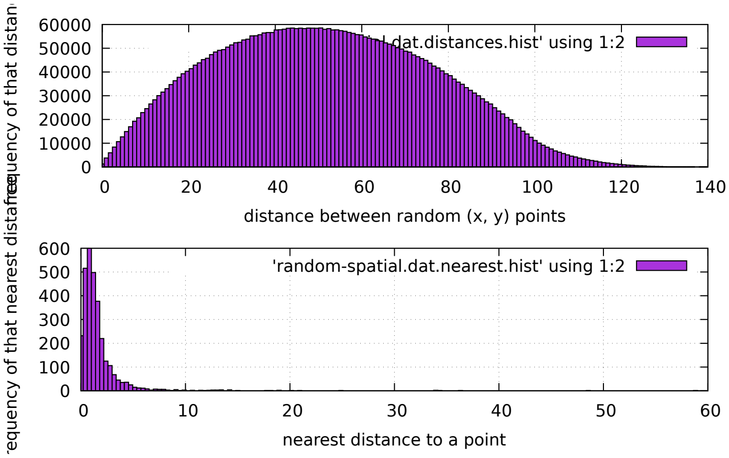


Figure 13.3.4: Histogram of distances between 2D random points.

Questions

So what would the two dimensional distribution look like if the (x, y) values were not purely random?

Explore this in two ways:

1. Change the program `random-spatial.py` to put all the (x, y) points at integer positions.
2. Change the program `random-spatial.py` to only put new points outside a "radius of avoidance" from existing points.

The "radius of avoidance" idea is what happens with the famous Waitomo Glowworm Caves in New Zeland: the worm larvae take positions on the ceiling of the cave and glow to attract prey. They avoid settling near other larvae to avoid cannibalism. The result is a distribution that does not have the same filaments and voids as a purely random distribution.

The lack of structure (filaments, voids…) in the cave was observed by biologist Stephen Jay Gould. He conjectured that the reason was this radius of avoidance. Physicist Ed Purcell carried out a calculation and visualization similar to ours to confirm Gould's conjecture.

[cite Steven Pinker for the glowworm example]

13.4 Brownian motion

In 1827 Scottish botanist Robert Brown looked at pollen grains in water and noticed that the pollen was moved around in the water, jiggling randomly. This was one of the significant discoveries that led to the acceptance of the molecular theory.

In 1905 Einstein explained this motion by showing that the pollen was being buffeted around by the motion of individual water molecules.

Today we know that fluids (liquids and gasses) are made of molecules which are in constant motion. They bump in to each other, into the walls of their container, and they also bump in to larger particles floating around (such as the grain of pollen that Brown observed).

Simulating this behavior is an interesting challenge: it would involve keeping track of a large number of gas molecules and seeing when they bump in to a large sphere, which would represent the grain of pollen.

Right now I leave that more detailed programming task as an exercise, and will point that a huge number of bumps (10^{14} collisions/second) by molecules which are going in all directions means that our grain of pollen will be moving in a sort of *random walk*, the same kind we discussed and visualized in Section **??**.

What we will do here is write some programs that simulate such random walks and then look at answering the question “where will I end up?” More specifically we ask “how far did I go?”

Let us start with one dimension. The program `multiple-walks.py` will take several walks in one dimension and write out the final arrival point and how far we went.

Listing 13.4.1: multiple-walks.py – Take several random walks in one dimension; print out the arrival points.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk

"""

import random
import math
import sys

def main():
    x = 0
    n_steps = 10000
    n_walks = 10
    if len(sys.argv) == 3:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    distances = []            # we'll store all the distances we found
    ## now run several random walks
    for i in range(n_walks):
        final_x = take_walk(x, n_steps)
        distance = math.fabs(final_x) # how far did we get?
        distances.append(distance)
        print('%d: %g  %g' % (i, final_x, distance))

def take_walk(x, n_steps):
    """take a simple 1D random walk"""
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
    ## return the final location
    return x

main()
```

Nice, but can we have a plot?

Listing 13.4.2: multiple-walks-plot.py – Take several random walks in one dimension; print out the arrival points and make a plot of the trajectory as a function of time.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk.  Keep track of the
path each walk took and plot it.
"""

import random
import math
import sys
import matplotlib.pyplot as plt

def main():
    x = 0
    n_steps = 10000
    n_walks = 10
    if len(sys.argv) == 3:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    distances = []            # we'll store all the distances we found
    ## prepare some plot parameters
    plt.xlabel('step')
    plt.ylabel('distance from origin')
    plt.grid(True)
    ## now run several random walks
    for i in range(n_walks):
        path = take_walk(x, n_steps)
        final_x = path[-1]
        distance = math.fabs(final_x) # how far did we get?
        distances.append(distance)
        print('%d: %g  %g' % (i, final_x, distance))
        plt.plot(path)        # add to the plot
    ## now show the plot we have accumulated
    plt.show()

def take_walk(x, n_steps):
    """take a simple 1D random walk"""
    path = []
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
        path.append(x)
    ## return the final location
```

(continues on next page)

(continued from previous page)

```
    return path

main()
```

Sure, but what if we are impatient and we want to see the plots before they are all generated. We can animate the plot using the techniques from Section **??**.

Listing 13.4.3: multiple-walks-plot.py – Take several random walks in one dimension; print out the arrival points and make a plot of the trajectory as a function of time. In this version we draw the plots as the walks become ready, rather than waiting for the end.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk.  Keep track of the
path each walk took and plot it.
"""

import random
import math
import sys
import matplotlib.pyplot as plt

def main():
    x = 0
    n_steps = 10000
    n_walks = 10
    if len(sys.argv) == 3:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    distances = []            # we'll store all the distances we found
    ## prepare to have animation
    fig, ax = plt.subplots()
    plt.xlabel('step')
    plt.ylabel('distance from origin')
    plt.grid(True)
    plt.ion()
    ## now run several random walks
    for i in range(n_walks):
        path = take_walk(x, n_steps)
        final_x = path[-1]
        distance = math.fabs(final_x) # how far did we get?
        distances.append(distance)
        print('%d: %g  %g' % (i, final_x, distance))
        plt.plot(path)        # add to the plot
        fig.canvas.draw_idle()
        plt.pause(0.4)
```

(continues on next page)

(continued from previous page)

```
        fig.canvas.draw_idle()
    ## now show the plot we have accumulated

    plt.show()
    plt.waitforbuttonpress()

def take_walk(x, n_steps):
    """take a simple 1D random walk"""
    path = []
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
        path.append(x)
    ## return the final location
    return path

main()
```

Now let's look at a crucial property of random walks: the overall behavior of the final position and the distance from the origin. At firs tit seems that these are all very different, with no pattern. But if we plot a *histogram* of the final positions in the one dimensional random walk we see something interesting:

Listing 13.4.4: multiple-walks-histogram.py – Take several random walks in one dimension; plot a histogram of the arrival points. If we take enough walks and they are long enough, we end up with a gaussian distribution of positions, centered around the origin.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk.  This version starts
from multiple-walks.py and adds the ability to plot a histogram of how
far we got.

"""

import random
import math
import sys

import numpy as np
import matplotlib.pyplot as plt

fig, ax = plt.subplots()

def main():
    x = 0
    n_steps = 1001
    n_walks = 1001
    if len(sys.argv) == 3:
```

(continues on next page)

(continued from previous page)

```
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    positions = []            # we'll store all the positions we found
    ## prepare to plot
    plt.grid(True)
    plt.xlabel('final value')
    plt.ylabel('number of walks')
    plt.title('1D random walk arrival values (%d, %d)' % (n_steps, n_walks))
    plt.ion()                   # enable interactive
    ## now run several random walks
    for i in range(n_walks):
        final_x = take_walk(x, n_steps)
        positions.append(final_x)
        # print('%d: %g  %g' % (i, final_x, position))
        plot_position_histogram(positions, n_steps, n_walks)
    plt.show()
    plt.waitforbuttonpress()


def take_walk(x, n_steps):
    """take a simple 1D random walk"""
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
    ## return the final location
    return x

def plot_position_histogram(positions, n_steps, n_walks):
    if len(positions) < 10:
        return
    n_steps_so_far = len(positions)
    n_bins = min(51, 5+int(n_steps_so_far/3))
    # print(np.array(positions))
    plt.cla()
    plt.grid(True)
    plt.xlabel('final value')
    plt.ylabel('number of walks')
    plt.title('1D random walk arrival values (%d, %d)'
              % (n_steps_so_far, n_walks))
    n, bins, patches = plt.hist(positions, n_bins)
    fig.canvas.draw_idle()
    plt.pause(0.01)

main()
```

NOTE: try to superimpose a gaussian with sigma proportional to sqrt(n_steps).

Let’s visualize things together with an animated multiplot:

Listing 13.4.5: multiple-walks-histogram-multiplot.py – This version combines the plotting of the 1D random walks and the histograms, and shows them as they accumulate.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk.  This version starts
from multiple-walks.py and adds the ability to plot a histogram of how
far we got.

"""

import random
import math
import sys

import numpy as np
import matplotlib.pyplot as plt

skip_interval = 1
# skip_interval = 5

def main():
    x = 0
    n_steps = 1001
    n_walks = 1001
    if len(sys.argv) == 3:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    positions = []            # we'll store all the positions we found
    # (ax1, ax2, ax3, ax4) = prepare_plotting(n_steps, n_walks)
    prepare_plotting(n_steps, n_walks)
    ## now run several random walks
    last_10_walks = []
    for i in range(n_walks):
        take_walk(x, n_steps, last_10_walks)
        if i % skip_interval == 0 or i < 20 or i > (n_walks-20):
            plot_walks(last_10_walks)
        final_x = last_10_walks[-1][-1] # last position of last walk
        position = math.fabs(final_x) # how far did we get?
        # positions.append(position)
        positions.append(final_x)
        # print('%d: %g  %g' % (i, final_x, position))
        if i % skip_interval == 0 or i < 20 or i > (n_walks-20):
            plot_position_histogram(positions, n_steps, n_walks)
        plt.subplot(224).figure.suptitle('1D random walks arrival values (%d, %d)'
                                         % (i, n_walks))
```

(continues on next page)

(continued from previous page)

```
        plt.pause(0.000001)
    plt.show()
    plt.waitforbuttonpress()

def prepare_plotting(n_steps, n_walks):
    fig = plt.figure(1)
    fig.suptitle('1D random walks arrival values (%d, %d)'
                 % (n_steps, n_walks))
    plt.subplots_adjust(top=0.92, bottom=0.10, left=0.10, right=0.95,
                        hspace=0.25, wspace=0.35)
    plt.ion()
    # return (ax1, ax2, ax3, ax4)


def take_walk(x, n_steps, last_10_walks):
    """take a simple 1D random walk"""
    last_10_walks.append([])
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        x = x + step_x
        last_10_walks[-1].append(x)
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
    ## now make sure we only keep the last 10 entries
    while len(last_10_walks) > 10:
        del last_10_walks[0]
    return last_10_walks

def plot_walks(last_10_walks):
    ## plot it in the cumulative plot
    ax = plt.subplot(2, 2, 1)
    plt.xlabel('step')
    plt.ylabel('position')
    ax.plot(last_10_walks[-1])
    ## plot it in the "last 10" plots
    if len(last_10_walks) >= 10:
        ax = plt.subplot(2, 2, 2)
        ax.cla()
        plt.xlabel('step')
        plt.ylabel('position')
        for i in range(10):
            ax.plot(last_10_walks[i])
    ax.figure.canvas.draw_idle()


def plot_position_histogram(positions, n_steps, n_walks):
    if len(positions) < 10:
        return
    n_steps_so_far = len(positions)
    n_bins = min(51, 5+int(n_steps_so_far/3))
    ## plot the positions
    ax = plt.subplot(2, 2, 3)
```

(continues on next page)

(continued from previous page)

```
    ax.cla()
    ax.grid(True)
    plt.xlabel('final position')
    plt.ylabel('number of walks')
    # plt.title('1D random walk (%d, %d)'
    #           % (n_steps_so_far, n_walks))
    n, bins, patches = ax.hist(positions, n_bins)
    ax.figure.canvas.draw_idle()
    ## plot the positions
    distances = [math.fabs(pos) for pos in positions]
    ax = plt.subplot(2, 2, 4)
    ax.cla()
    ax.grid(True)
    plt.xlabel('distance from start')
    plt.ylabel('number of walks')
    # plt.title('1D random walk (%d, %d)'
    #           % (n_steps_so_far, n_walks))
    n, bins, patches = ax.hist(distances, n_bins)
    ax.figure.suptitle('1D random walks arrival values (%d, %d)'
                       % (n_steps_so_far, n_walks))
    ax.figure.canvas.draw_idle()

main()
```

Let's look at the 2D situation:

Listing 13.4.6: multiple-walks-histogram-2d.py – Take several random walks in two dimensions; plot a histogram of how far they arrived.

```
#! /usr/bin/env python3

"""Take many one-simensional random walks so that we can study the
distribution of how far you get in a random walk.  This version starts
from multiple-walks-histogram.py, changes it to take two dimensional
walks, and then plots a histogram of how far we got.

"""

import random
import math
import sys

import numpy as np
import matplotlib.pyplot as plt

def main():
    pt = (0, 0)
    n_steps = 1000
    n_walks = 1000
    if len(sys.argv) == 3:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
```

(continues on next page)

(continued from previous page)

```
    elif len(sys.argv) == 4:
        n_steps = int(sys.argv[1])
        n_walks = int(sys.argv[2])
        output_file = sys.argv[3]
    elif len(sys.argv) == 2:
        output_file = sys.argv[1]
    elif len(sys.argv) != 1:
        sys.stderr.write('error - usage: %s [n_steps n_walks]\n' % sys.argv[0])
        exit(1)

    distances = []            # we'll store all the distances we found
    ## now run several random walks
    for i in range(n_walks):
        final_pt = take_walk(pt, n_steps)
        distance = math.hypot(*final_pt) # how far did we get?
        distances.append(distance)
        # print('%d: %g  %g  %g' % (i, *final_pt, distance))
    plot_distance_histogram(distances, n_steps, n_walks, output_file)

def take_walk(pt, n_steps):
    """take a simple 1D random walk"""
    for i in range(n_steps):
        ## generate a step of -1 or +1 in the x direction
        step_x = random.randint(0, 1) * 2 - 1
        step_y = random.randint(0, 1) * 2 - 1
        pt = (pt[0] + step_x, pt[1] + step_y)
        # print(i, x, math.sqrt(i), math.fabs(math.fabs(x) - math.sqrt(i)))
    ## return the final location
    return pt

def plot_distance_histogram(distances, n_steps, n_walks, output_file):
    n_bins = min(50, n_walks/3.0)
    # print(np.array(distances))
    n, bins, patches = plt.hist(distances, n_bins)
    print(bins)
    plt.xlabel('distance traveled')
    plt.ylabel('number of walks')
    plt.title('2D random walk distance distribution (%d, %d)'
              % (n_steps, n_walks))
    plt.grid(True)
    plt.show()

main()
```

13.5 Further reading

The wikipedia article https://en.wikipedia.org/wiki/Brownian_motion has nice diagrams.

Start by watching this video: https://www.youtube.com/watch?v=hy-clLi8gHg

Possibly do this experiment: https://www.youtube.com/watch?v=wf2tBAvMNbg

Other video on brownian motion: https://www.youtube.com/watch?v=NHo6LTXdFns

This one has a cute demonstration of how you get a random walk when you focus on a single particle, and is https://www.youtube.com/watch?v=pdz7wFHSLD0

Scipy has some routines that help simulate Brownian motion. The cookbook has this example: http://scipy-cookbook.readthedocs.io/items/BrownianMotion.html

13.6 Progression of record peaks

http://iaaf-ebooks.s3.amazonaws.com/2015/Progression-of-IAAF-World-Records-2015/projet/IAAF-WRPB-2015.pdf

https://en.wikipedia.org/wiki/Athletics_record_progressions

[footnotes]

13.7 The gambler's fallacy

13.8 The gambler's ruin

13.9 Link to other chapters

- Simulated annealing covered in Section **??**
- Genetic algorithms (planned).

CHAPTER

FOURTEEN

POWER LAWS, ZIPF, BENFORD, …

Areas: pure math, curiosity, economics, complex systems

[status: partly written]

14.1 Motivation, prerequisites, plan

We can start by realizing that certain relationships between numbers are “curious”. Our curiosity might be enough to get us going if we see a couple of examples. Here are two examples which might make us curious:

- Longer words in a language occur less often, and there is a clear mathematical relationship between frequency of words and the rank of the word in a frequency table (Zipf’s law).
- In numbers that come up in the real world, the most significant digit is more often 1 and 2 than 8 and 9 (Benford’s law).

There are many areas of nature and of human endeavor in which similar relationships come up. These relationships are called *power laws* (where *power* refers to the exponent in a variable). Many of these give insight into the underlying mechanisms of those areas of study.

We will start by studying Zipf’s law and write some programs that illustrate it. This will let us study other data sets that have similar relationships, some from astronomy, some from geography, some from economics. We will look at those examples, discussing the insights that come from the power law relationships. Finally we will dedicate some time to Benford’s law and some curious and amusing historical applications.

14.2 A brief refresher on log-log plots

Let us try plotting a power law function in gnuplot:

```
gnuplot> set samples 1000
gnuplot> plot x**(-1)
gnuplot> plot [0.01:100] x**(-1)
```

```
gnuplot> set samples 1000
gnuplot> a = 1
gnuplot> k = -1.5
gnuplot> plot [0.01:100] a * x**k
gnuplot> k = -3.2
gnuplot> replot [0.01:100] a * x**k
```

Note how little information you actually see about the plot. That's because the scales of the information for x near 0 and for x far out on the x axis are very different. You can zoom in with the gnuplot GUI (right-click and drag a rectangle near the origin), but that doesn't tell you much more.

We usually take the logarithms of both x and y and then we can see the nuances of what's happening in between:

```
gnuplot> set logscale xy
gnuplot> set samples 1000
gnuplot> a = 1
gnuplot> k = -1.5
gnuplot> plot [0.01:100] a * x**k
gnuplot> k = -3.2
gnuplot> replot [0.01:100] a * x**k
```

Note that the slope of these "log-log" plots is that exponent k.

14.3 Zipf's law

We start with Zipf's law. In the first half of the 20th century George Zipf noticed that if you have a large sample of words in a real-world English language text, there is an inverse proportionality between the frequency of a word and its position in the frequency table (see Section **??** where we introduced inverse proportionality).

This means that a plot of frequency versus rank will look like a $1/x$ plot, or if you plot both x and y axis on a logarithmic scale, you will have a straight line with a slope of -1.

How do we explore this law? As usual, we write a program! Start by typing in the program in Listing **??**.

Listing 14.3.1: word-freq-rank.py - Analyze frequency versus rank for words in a text file.

```
#! /usr/bin/env python3

"""
Reads all the words in a file and prints information about the
rank and frequence of occurrence of words in the file.

The file should be a rather long file with a typical sampling of
words.  The ideal file would be a book downloaded from Project
Gutenberg in ascii text format.
"""

import sys
import re

def main():
    if len(sys.argv) == 1:      # handle command-line arguments
        f = sys.stdin
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        sys.stderr.write('error - usage: %s [filename]\n' % sys.argv[0])
        sys.exit(1)
```

(continues on next page)

(continued from previous page)

```
    sorted_words, word_freq_map = read_words_from_file(f)
    f.close()
    print('##  file:', fname)
    print('##  rank  word            frequency')
    for i, word in enumerate(sorted_words):
        print('%8d  %-13s  %8d' % (i+1, word, word_freq_map[word]))

def read_words_from_file(f):
    """read the words from a file and return two things: the sorted
    list of words, and the rank dictionary which maps each word to its
    rank."""
    word_set = set()
    word_freq_map = {}
    for line in f.readlines():
        word_list = re.split('--|\s', line)
        ## now that we have the words, let's strip away all the
        ## punctuation marks
        word_list = [word.strip(""",.;:_"'&%^$#@!?/\|+-()*""").lower()
                     for word in word_list]
        cleaned_up_words = []
        for word in word_list:
            word = word.strip(""",.;:_"'&%^$#@!?/\|+-()*""")
            if len(word) > 0:
                cleaned_up_words.append(word)
        ## now that we have found the words in this line, we also add
        ## them to the word rank dictionary before we lost the count
        ## information by adding them to the set
        for word in word_list:
            if word in word_freq_map.keys():
                word_freq_map[word] += 1
            else:
                word_freq_map[word] = 1
        ## finally, add them to a set, which discards repeated
        ## occurrences
        word_set.update(tuple(cleaned_up_words))

    ## finally: use the rank dictionary to sort the list of words by
    ## how often they occur (their rank)
    sorted_word_list = sorted(list(word_set), key=lambda x: -word_freq_map[x])
    return sorted_word_list, word_freq_map


main()
```

14.3.1 Example from a paragraph of your own.

Type a simple paragraph of English text into a file, call it *simple-para.txt*. Then run the program on that file with:

```
$ python3 word-freq-rank.py simple-para.txt
```

You'll see output showing the histogram with ASCII output.

You could also have run the program with no arguments and typed directly in the terminal.

14.3.2 Example from Project Gutenberg

To download a full book from Project Gutenberg and plot its word frequency distribution with gnuplot you can use these instructions:

Listing 14.3.2: Instructions to plot the word frequency distribution of Swann's Way, by Marcel Proust.

```
##REQUIRED_FILE: swanns-way.txt
##PRE_EXEC: wget --output-document swanns-way-english.txt http://www.gutenberg.org/cache/
↪epub/1128/pg1128.txt
##PRE_EXEC: ./word-length-freq.py swanns-way-english.txt > swanns-way-freq-english.out
set multiplot layout 2,1 title "Zipf's law"
set grid
plot 'swanns-way-freq-english.out' using 1:3 with linespoints pt 6 ps 0.2 title "word␣
↪frequency (linear scale)"
set logscale xy
plot 'swanns-way-freq-english.out' using 1:3 with linespoints pt 6 ps 0.2 title "word␣
↪frequency (log-log scale)"
```

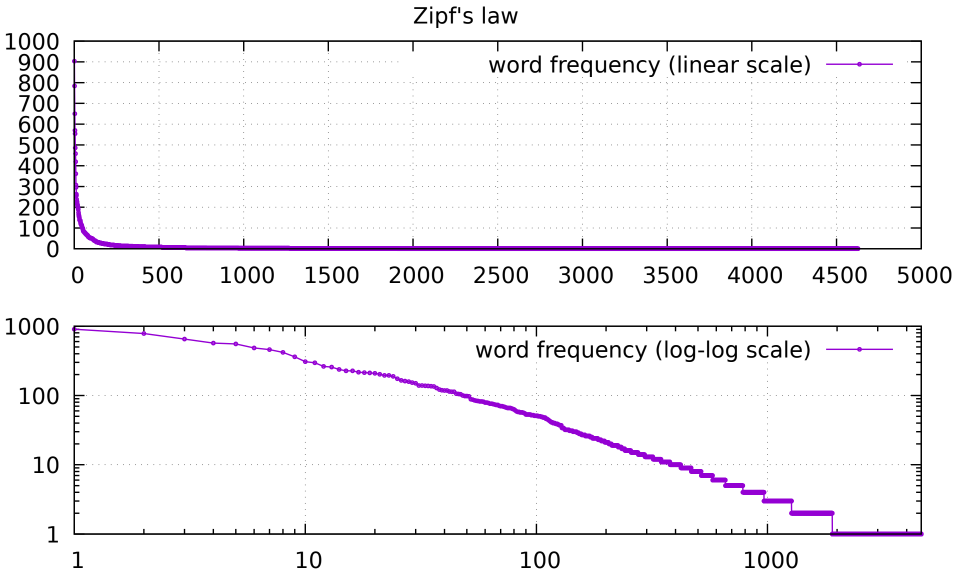


Figure 14.3.1: Histogram of how many times words of a certain length appear in the text of Swann's Way (volume 1 of Marcel Proust's "In Search of Lost Time", formerly translated in English as "Remembrance of Things Past").

The top plot in Figure **??** is in the usual linear scale. This can be hard to read because you have some big numbers, and the small ones are hard to tell apart. That’s why the other two plots have a *logarithmic* scale for the y axis (log scales are discussed in Section **??**).

Out of curiosity, let us also look at the first few and last few lines of the output file.

Listing 14.3.3: First few lines (most common words) in Swann’s Way. First column is the rank (most frequent to least frequent), the second is the word, the third is the number of times it occurs.

```
##  file: swanns-way-english.txt
##  rank  word            frequency
       1  the                 903
       2  and                 784
       3  i                   650
       4  to                  570
       5  of                  554
       6  you                 486
       7  my                  458
       8  a                   419
```

Listing 14.3.4: Last few lines (least common words) in Swann’s Way. First column is the rank (most frequent to least frequent), the second is the word, the third is the number of times it occurs.

```
    4616  ballow                1
    4617  starv'd               1
    4618  derive                1
    4619  official              1
    4620  succeed               1
    4621  cities                1
    4622  lately                1
    4623  whatever              1
    4624  simp'ring             1
    4625  constrains            1
    4626  sue                   1
    4627  vor                   1
```

Now let’s look at a comparison of the English translation of this book versus the original French text:

Listing 14.3.5: Instructions to plot the word frequency distribution of Swann’s Way, by Marcel Proust. This version compares the original French text with the English translation.

```
##REQUIRED_FILE: swanns-way-french.txt
##REQUIRED_FILE: swanns-way-english.txt
##PRE_EXEC: wget --output-document swanns-way-english.txt http://www.gutenberg.org/cache/
↪epub/1128/pg1128.txt
##PRE_EXEC: wget --output-document swanns-way-french.txt https://www.gutenberg.org/files/
↪2650/2650-0.txt
##PRE_EXEC: ./word-length-freq.py swanns-way-english.txt > swanns-way-freq-english.out
##PRE_EXEC: ./word-length-freq.py swanns-way-french.txt > swanns-way-freq-french.out
set grid
set ylabel 'word frequency'
set xlabel 'word rank'
```

(continues on next page)

(continued from previous page)

```
set logscale xy
set title "rank-frequency relationship for Swann's Way, comparing French and English"
plot 'swanns-way-freq-english.out' using 1:3 with linespoints, 'swanns-way-freq-french.
↪out' using 1:3 with linespoints
```

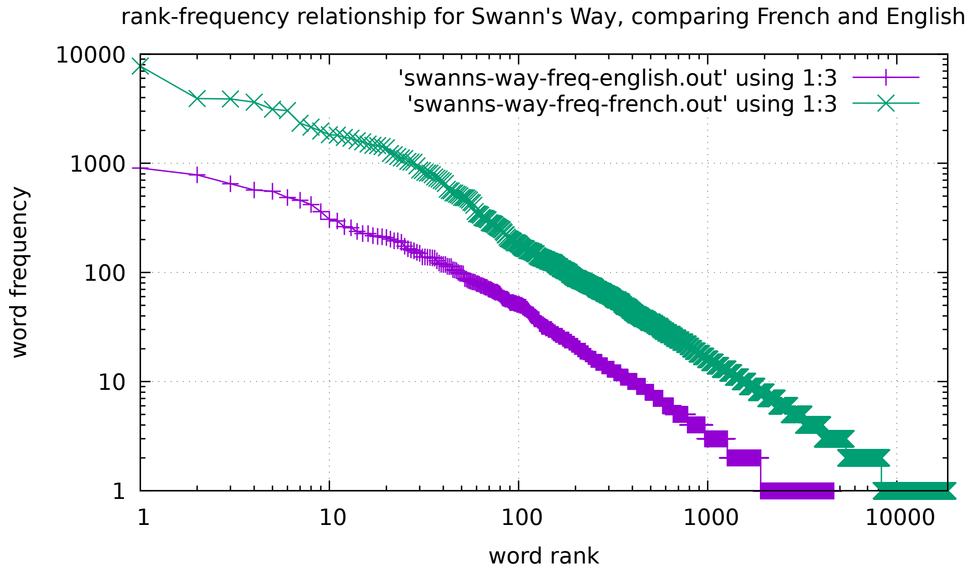


Figure 14.3.2: Histogram of how many times words of a certain length appear in the text of Swann's Way (volume 1 of Marcel Proust's "In Search of Lost Time", formerly translated in English as "Remembrance of Things Past"). This version compares the original French text with the English translation. Note that the slope for the main part of the curve is the same.

14.3.3 Explanations of Zipf's law

TODO

14.4 What are "power laws"?

If you have a relationship where the number of items with a given measure is proportional to that measure to a negative power, then we say that we have *power law* behavior.

In the case of Zipf's law, if l is the length of the words and $N(l)$ is the number of words with that length in our text, we have:

$$N(l) = \mathrm{const} \times l^{-1}$$

More in general we will have:

$$N(x) = \mathrm{const} \times x^{-\alpha}$$

where α is the "power" in "power law". In Zipf's law the power is $\alpha = -1$.

14.4.1 Calculating the power from the data

How do you calculate the power α if you are given a collection of data points?

In Section **??** we learned some techniques to fit functions to data. How would we carry out a fit to the data that brings out the power in the power law?

Zipf's law was just one rather notable example of power law, but there are many others. We will discuss a few more and write programs that download the relevant data and analyze it.

- population-rank distribution for nations

wget https://raw.githubusercontent.com/datasets/population/master/data/population.csv

- easier:

wget https://www.cia.gov/library/publications/the-world-factbook/rankorder/rawdata_2119.txt

- rank-size distribution for nations
- city size
- deadly conflicts
- luminosity in astronomy

14.5 Deadly conflicts

https://github.com/zacharykitt/analyses/blob/master/deadly_quarrels.py

14.6 Benford's law

14.6.1 Physical constants

```
$ wget https://physics.nist.gov/cuu/Constants/Table/allascii.txt
$ cat allascii.txt | cut -c 61 | grep '[0-9]' > first-digits.dat
gnuplot> plot 'first-digits.dat'
```

This has downloaded an ascii table of physical constants (we've talked about ascii before, but we need to frequently remind people that ascii is basically plain text… ascii is not a common buzzword, so we should mention ascii often, and explain often what it means).

Now we want to look at the distribution of those first digits. Which appears most?

In Section **??** we learned how to make and plot histograms. The most effective way to make and plot histograms might be the snippet of code that uses numpy and matplotlib in Section **??**. Let's use that to write a small program that analyzes the frequency of these first digits:

Listing 14.6.1: first-digit-hist.py - Plot the frequency with which digits appear in the first position in a list of numbers.

```
#! /usr/bin/env python3

## run this with
## ./first-digit-hist.py first-digits.dat
## to get an interactive plot, or with
```

(continues on next page)

(continued from previous page)

```
## ./first-digit-hist.py first-digits.dat first-digit-freq.png
## (or you can put pdf or svg)

import sys
import numpy as np
import matplotlib.pyplot as plt

fname = sys.argv[1]
out_fname = ''
if len(sys.argv) == 3:
    out_fname = sys.argv[2]
    out_format = sys.argv[2][-3:]

## [...] collect quantity to be histogrammed in an array x
xs = open(fname, 'r').read().split()
x = [int(digit) for digit in xs]
## prepare the plot
n, bins, patches = plt.hist(x, bins=np.arange(1,11)-0.5,
                            density=1, facecolor='g', alpha=0.75)
plt.xticks(range(1,10))
plt.xlim([0,10])
plt.xlabel('first digit')
plt.ylabel('probability')
plt.title('Histogram of first digit probability')
plt.grid(True)
## if there is no output filename then we plot interactively
if not out_fname:
    plt.show()
else:
    plt.savefig(out_fname, format=out_format)
    print('output written to file %s' % out_fname)
```

Go ahead and enter the program `first-digit-hist.py` and then run it with:

```
## to get an interactive plot:
$ ./first-digit-hist.py first-digits.dat
## to put the output into a file:
$ ./first-digit-hist.py first-digits.dat first-digit-freq.png
## or you can replace .png with .pdf or .svg
```

The result of these runs should be a plot a bit like the one in Figure **??**: you will see that 1 occurs much more frequently, followed by 2 and then 3. Once you get beyond 4 you would have to collect a lot of data to get a clear result, but you would then find that 4 occurs more than 5, which occurs more than 6 and so on.

To show an example of how one can use shell pipelines to look at another data set. The GNU scientific library sources can be found on github at

```
$ wget https://raw.githubusercontent.com/ampl/gsl/master/const/gsl_const_mks.h
$ wget http://git.savannah.gnu.org/cgit/gsl.git/plain/const/gsl_const_cgs.h
$ cat gsl_const_mks.h | grep 'define GSL_' | awk '{print $3}' | grep -v '(1e' | cut -c 2 > first-digits-gsl-mks.dat
$ cat gsl_const_cgs.h | grep 'define GSL_' | awk '{print $3}' | grep -v '(1e' | cut -c 2 > first-digits-gsl-cgs.dat
```

(continues on next page)

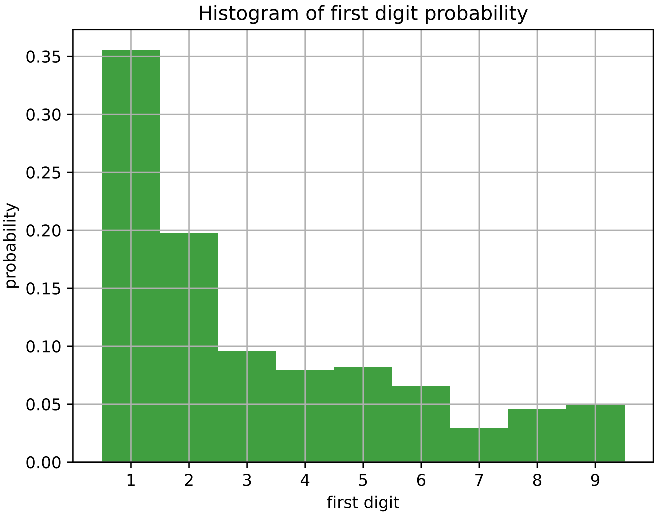


Figure 14.6.1: Histogram of how many times the various digits appear as the *first* digit in a number. The list of numbers we use are the physical constants in the National Institute of Standards and Technology list.

(continued from previous page)

```
$ ./first-digit-hist.py first-digits-gsl-cgs.dat &
$ ./first-digit-hist.py first-digits-gsl-mks.dat &
```

14.6.2 Stock quotes

Another source of numbers to which Benford's law might apply is stock quotes. Let us FIXME

```
wget --output-document nasdaq-list.csv 'http://www.nasdaq.com/screening/companies-by-
↪industry.aspx?exchange=NASDAQ&render=download'
```

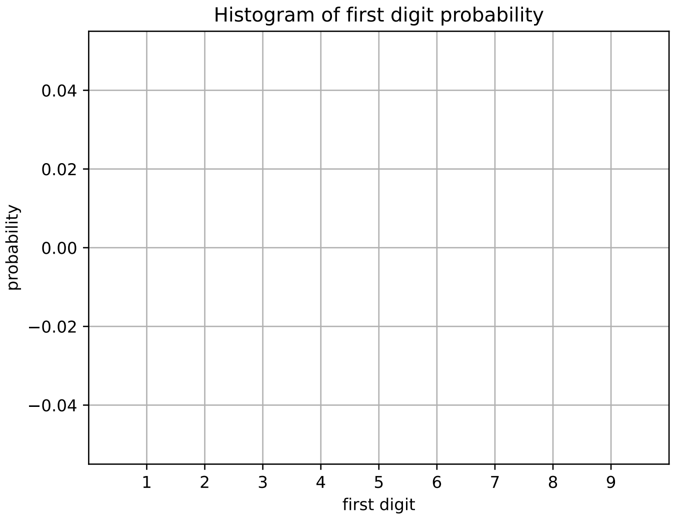
Listing 14.6.2: first-digit-nasdaq.py - Plot the frequency with which digits appear in the first position of NASDAQ stock market listings.

```
#! /usr/bin/env python3

## run this with
## ./first-digit-hist.py first-digits.dat
## to get an interactive plot, or with
## ./first-digit-hist.py first-digits.dat first-digit-freq.png
## (or you can put pdf or svg)

import sys
import numpy as np
import matplotlib.pyplot as plt
import csv

fname = sys.argv[1]
out_fbase = ''
```

(continues on next page)

(continued from previous page)

```
if len(sys.argv) == 3:
    out_fbase = sys.argv[2][:-4]
    out_format = sys.argv[2][-3:]

print(out_fbase)

csvfile = open(fname, 'r')
reader = csv.reader(csvfile, delimiter=',', quotechar='"')
firstdigit_sale = []
firstdigit_cap = []

for row in list(reader)[1:]:
    print(row[0], row[2], row[3])
    if row[2] != 'n/a':
        sale = float(row[2])
        if sale != 0:
            firstdigit = ('%e' % sale)[0]
            firstdigit_sale.append(int(firstdigit))
    if row[3] != 'n/a':
        cap = float(row[3])
        if cap != 0:
            firstdigit = ('%e' % cap)[0]
            firstdigit_cap.append(int(firstdigit))
csvfile.close()

## prepare the plot: collect quantity to be histogrammed in an array x
for (label, x) in [('sale', firstdigit_sale), ('cap', firstdigit_cap)]:
    n, bins, patches = plt.hist(x, bins=np.arange(1, 11)-0.5,
                                density=1, facecolor='g', alpha=0.75)
    plt.xticks(range(1, 10))
    plt.xlim([0, 10])
    plt.xlabel('first digit')
    plt.ylabel('probability')
    plt.title('Histogram of first digit probability')
    plt.grid(True)
    ## if there is no output filename then we plot interactively
    if not out_fbase:
        plt.show()
    else:
        out_fname = out_fbase + '_' + label + '.' + out_format
        plt.savefig(out_fname, format=out_format)
        plt.gcf().clear()
        print('output written to file %s' % out_fname)
```



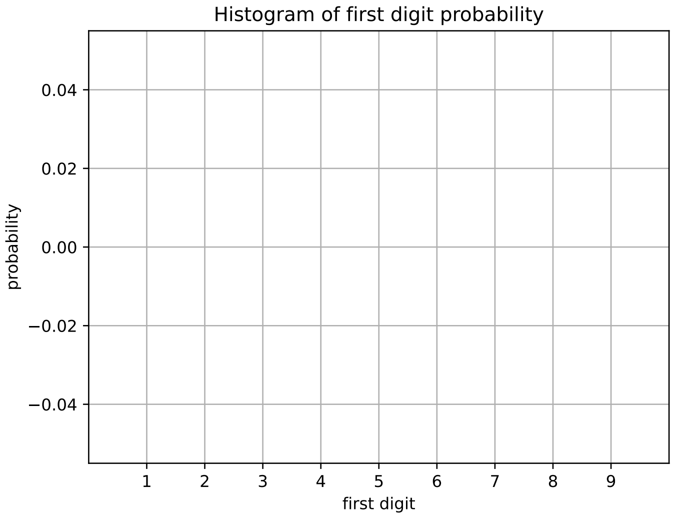


Figure 14.6.2: Histogram of how many times the various digits appear as the *first* digit in a number. The list of numbers we use here are the stock quotes and market capitalization in the NASDAQ stock exchange.

14.6.3 Further reading

- https://physics.nist.gov/cuu/Constants/Table/allascii.txt
- https://en.wikipedia.org/wiki/Benford%27s_law
- https://en.wikipedia.org/wiki/Zipf%27s_law
- stats on word length
- /usr/share/dict/words
- https://en.wikipedia.org/wiki/Brown_Corpus
- http://www.nltk.org/nltk_data/
- random book from gutenberg
- stats on individual income or wealth
- War and conflict datasets: https://ucdp.uu.se/downloads/
- Correlates of war study: https://en.wikipedia.org/wiki/Correlates_of_War
- Correlates of war data: http://correlatesofwar.org/data-sets/COW-war
- Richardson's original data set: https://vincentarelbundock.github.io/Rdatasets/doc/HistData/Quarrels.html
- Detailed paper with many war diagrams: https://arxiv.org/pdf/1901.05086.pdf https://pdfs.semanticscholar.org/27cd/75962a447881768698ac1402100938d5f790.pdf

14.7 Pareto's principle

14.7.1 Further reading

- https://www.theguardian.com/commentisfree/2011/nov/11/occupy-movement-wealth-power-law-distribution

14.8 Olber's paradox

CHAPTER

FIFTEEN

PUSHING TOWARD CALCULUS

[status: somewhat written but incomplete]

15.1 Motivation and plan

When I was in school calculus is where math became a pure and never-ending thrill.

Knowing calculus will get us on a path toward science the way it is really done: scientific laws describe how biological or chemical or physical systems change in time and position, and calculus is the mathematics of those changes.

So introducing calculus opens up a fantastic world to us, but how do we teach it to middle and high school kids who have not yet learned the prerequisites?

Our advantage over a pure math class is that we can write programs to illustrate calculus topics. We might not yet know trigonometric and logarithmic identities, but we can write programs to convince ourselves of the properties we have not yet proven. The same goes for limits. This means that we can jump ahead for a while, knowing that in our pure math classes we will eventually prove all the material that we now understand intuitively.

15.2 Prerequisites

- The 10 hour programming workshop.
- The mini-courses on elementary and intermediate plotting in Section **??** and Section **??**
- The "tour of functions" mini-course in Section **??**.

15.3 Limits, the infinitely big, and the infinitesimally small

Let us plot a function: $f(x) = 1/x$:

```
gnuplot> set grid
gnuplot> plot 1/x
```

We see that it spikes both down and up when x is close to zero, and that it gets closer and closer to zero as x gets very big in the positive direction, and also when x gets very big in the negative direction.

This brings up some questions:

1. Does $f(x)$ ever actually reach zero when x gets big?
2. What is $f(0)$? We cannot divide by zero, so the answer cannot be $f(0) = 1/0$.

There is no clear answer to these questions, but as we begin to grope for answers we might want to say something like:

1. $f(x)$ will reach zero when x is infinity, or minus infinity. In formulae: $f(\infty) = 1/\infty = 0$ and $f(-\infty) = 1/(-\infty) = 0$.
2. $f(0)$ is infinite. So: $f(0) = 1/0 = \infty$.

But there are several problems with writing down these expressions.

First of all infinity is not an actual number, so we do not have a proper arithmetic for it. We have to define what all of that means.

Then there is the obvious worry that from the plot it looks like $f(0) = \infty$ (when you look from the right) *and* $f(0) = -\infty$ (when you look from the left). This is not OK becuase a function can only have one value at a given point.

The way mathematicians tackle this is with the idea of a *limit*. Let us start by using this kind of terminology for our $1/x$ example. For the case of x getting big:

"The limit of $1/x$ as x goes to infinity is zero." (And "The limit of $1/x$ as x goes to minus infinity is zero.")

And the way we define that is (taking some liberty):

"You can make $1/x$ arbitrarily close to zero by making x big enough."

A more formal way we can say it is:

"For any number (no matter how small) ϵ, I can find a value x_0 such that, for all $x > x_0$, $f(x) < \epsilon$."

So that allows us to talk about the behavior of $1/x$ as x gets really big (or really big in the negative direction): we do not say that $1/\infty = 0$, but rather we say that the limit of $1/x$ is 0 as x gets really big. The math notation for this is:

$$\lim_{x \to \infty} \frac{1}{x} = 0$$

although one can also say:

$$\frac{1}{x} \to 0 \text{ as } x \to \infty$$

And how do mathematicians deal with $f(0)$? That's the one where we have different behavior to the left and to the right of $x = 0$.

For this mathematicians define a *one-sided limit*. The way we can talk about $1/x$ around 0 is:

"The limit of $1/x$ as x goes to 0 *from above* is infinity."

$$\lim_{x \to 0^+} \frac{1}{x} = \infty$$

and

"The limit of $1/x$ as x goes to 0 *from below* is minus infinity."

$$\lim_{x \to 0^-} \frac{1}{x} = -\infty$$

Note: Limits also apply to ordinary functions at ordinary points, not just to unusual situations. For example, $\lim_{x \to 3} x^2$ is simply $3^2 = 9$.

15.4 Continuous functions

15.5 Convergence and divergence

We saw that $f(x) = 1/x$ approaches zero as x goes to infinity. We say that $f(x)$ *converges* to zero as x goes to infinity.

We saw that $f(x) = 1/x$ goes wild at zero. We say that $f(x)$ *diverges* as x goes to zero.

Another example clarifies what mathematicians mean by divergence. Take this limit:

$$\lim_{x \to \infty} \sin(x)$$

The $\sin(x)$ function will bounce around forever, bounded between -1 and +1 forever - it will never settle close to a single value. We say that this function diverges, even though it's not blowing up to infinity or minus infinity: the fact that it does not converge to a single value means that the limit diverges.

15.6 Weird mixes

Our examples of $1/0$ and $1/\infty$ are straightforward, but what if you have a more complex limit, like:

$$\lim_{x \to 0} \sin(1/x)$$

Let us plot this one, and then zoom in to see if it converges:

```
gnuplot> set samples 1000
gnuplot> plot sin(1/x)
gnuplot> plot [-3:3] sin(1/x)
gnuplot> plot [-1:1] sin(1/x)
gnuplot> plot [-0.5:0.5] sin(1/x)
gnuplot> plot [-0.1:0.1] sin(1/x)
gnuplot> plot [-0.05:0.05] sin(1/x)
gnuplot> plot [-0.01:0.01] sin(1/x)
gnuplot> plot [-0.005:0.005] sin(1/x)
gnuplot> plot [-0.001:0.001] sin(1/x)
```

We see that there is no convergence, and in fact as you get closer to zero the function looks like it's *less* likely to converge on a single value.

Now let us look at:

$$\lim_{x \to 0} x \sin(1/x)$$

and let us plot this slighty different function in the same way:

```
gnuplot> set samples 1000
gnuplot> plot x*sin(1/x)
gnuplot> plot [-3:3] x*sin(1/x)
gnuplot> plot [-1:1] x*sin(1/x)
gnuplot> plot [-0.5:0.5] *sin(1/x)
gnuplot> plot [-0.1:0.1] x*sin(1/x)
gnuplot> plot [-0.05:0.05] x*sin(1/x)
gnuplot> plot [-0.01:0.01] x*sin(1/x)
gnuplot> plot [-0.005:0.005] x*sin(1/x)
gnuplot> plot [-0.001:0.001] x*sin(1/x)
```

Or let us automate that:

```
reset
set samples 5000
set yrange [-1:1]

do for [ii=1:100:1] {
    max = 100.0/(ii*ii)
    print(max)
    plot [-max:max] x*sin(1/x)
    pause 0.03
}
```

This zooming in shows us that the function $f(x) = x\sin(1/x)$ does not diverge at $x = 0$, even though it has that $1/x$ bit inside. Although we cannot take $f(0)$, we do have a limit that converges:

$$\lim_{x\to 0} x\sin(1/x) = 0$$

15.7 Limits of some functions

Listing 15.7.1: explore-limit.py – exploring the limiting behavior of some functions.

```
#! /usr/bin/env python3

def f(x):
    # return 1/x
    return 1/x**2
    # return exp(-x)
    # return exp(-x**2)

epsilon_str = input('give a very small number: ')
epsilon = float(epsilon_str)

x = 1
while f(x) >= epsilon:
    x += 1

print('with the value of x = %g we got f(x) < %g' % (x, epsilon))
```

15.8 The limit of a series

A *summation* is the addition of a sequence of numbers. We use the cool gree capital sigma letter Σ as a notation for this. For example, if we want to write Gauss's formula for the sum of the first 100 numbers:

$$\sum_{k=1}^{100} k = (100 * 101)/2 = 5050$$

More generally:

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Or the sum of the first 50 values of the *harmonic series* $1/k$:

$$\sum_{k=1}^{50} \frac{1}{k}$$

Or the sum of the first n values of $1/k^2$:

$$\sum_{k=1}^{n} \frac{1}{k^2}$$

We read this as "The sum of one over k squared with k going from one to n . . ."

Sometimes we have something called an *infinite series*, or just *series*. This is a sum where you keep adding up the elements forever. An example is:

$$\sum_{k=1}^{\infty} \frac{1}{k^2}$$

which looks like the previous example of a finite sum, but where we sum forever.

One question you might ask is "wait, if you add infinitely many things, won't that diverge?" My first answer is to say that I'm delighted at how you used the word "diverge" so comfortably. The answer to the question is that we are summing an infinite number of terms, but as we add more terms and k gets bigger, $1/k^2$ gets infinitesimally small! That means that it's *possible* that the sum will converge.

Let us write a simple program which tests this:

Listing 15.8.1: explore-series.py - explore whether the infinite sums of $1/k$ and $1/k^2$ diverge or converge.

```
#! /usr/bin/env python3

## compare the infinite sums of 1/k and 1/k^2

N = 500
sum_1_over_k = 0
sum_1_over_k_sq = 0

for k in range(1, N+1):
    sum_1_over_k += 1.0/k
    sum_1_over_k_sq += 1.0/(k*k)
    print(k, '    ', sum_1_over_k, '    ', sum_1_over_k_sq)
```

A cute fact to note is that:

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

so you should take a moment to take the result of the third column in the output of `explore-series.py`, multiply by 6, take the square root, and see if you get π.

We have seen two examples of infinite series, one of which converges and the other does not. Note that mathematicians have dozens of interesting ways of proving *rigorously* that the harmonic series diverges. A proof dating from the middle ages is shown in the Wikipedia article on the harmonic series

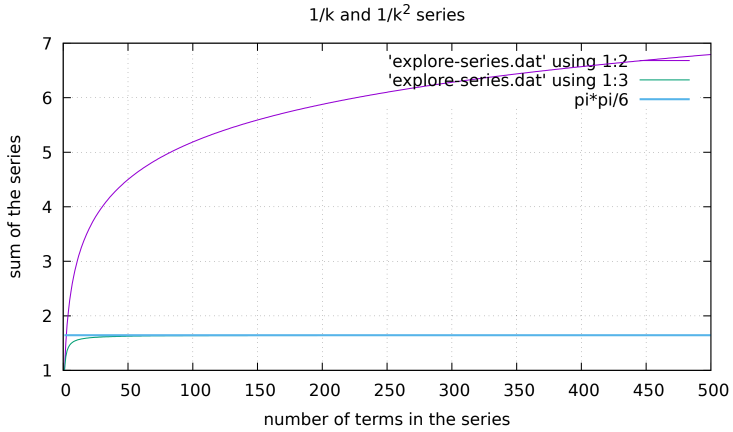


Figure 15.8.1: The sum of the sums of $1/k$ and $1/k^2$. We see that the sum of $1/k$ gets smaller but eventually does surpass any point you might pick, while the sum of $1/k^2$ never grows past $\frac{\pi^2}{6}$.

15.9 The most important application: derivatives

The most important use we have for limits is to formulate differential and integral calculus.

Differential calculus is the study of the *rate of change* of functions, while integral calculus is the study of the areas defined by the curves of functions.

We will discuss integrals in Section **??**, while we will talk briefly about derivatives here. [FIXME: eventually derivatives might also have their own chapter.]

There are many introductions to calculus, including some written as free documentation, so here I will simply show a couple of figures from the Wikipedia article on derivatives so that an instructor can make reference to those pictures and explain it on a blackboard or whiteboard.

In Figure **??** we see that a reasonably smooth curve has a tangent line at each point, and the slope of that tangent line (which we know from analytical geometry) is what we call the slope of the curve.

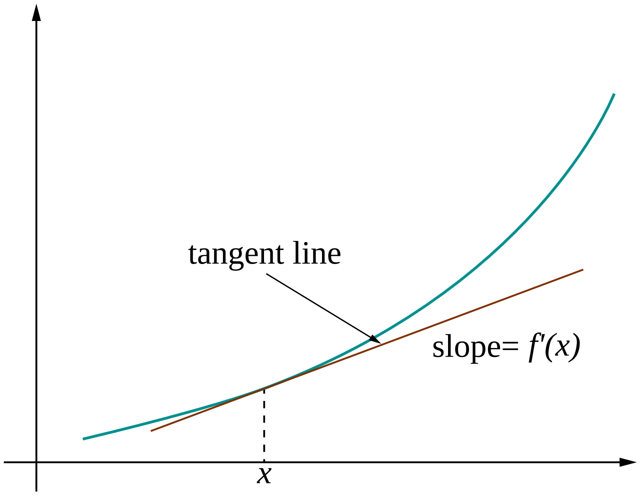


Figure 15.9.1: The derivative is slope of the the line tangent to the curve at the given point.

But how do we find the slope of this tangent line? In Figure **??** we see how we might do this:

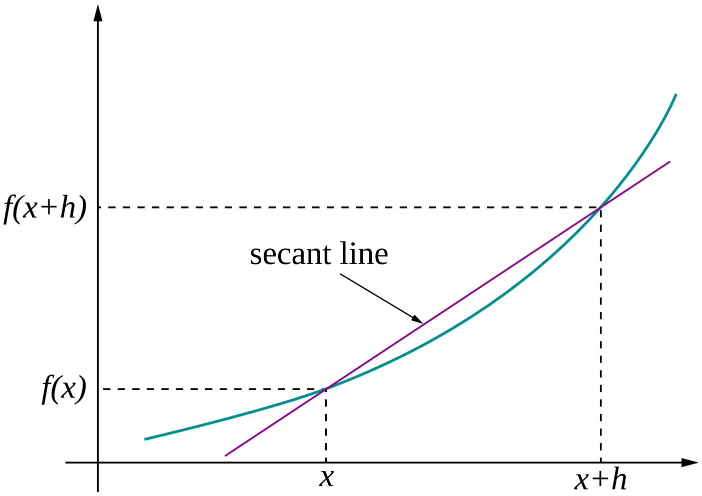


Figure 15.9.2: To find the tangent line we start with an *intersecting* (or *secant*) line that goes through the point x and a point a tiny bit ahead $x + h$, where h is small.

If we take the points $(x, f(x)), (x + h, f(x + h))$ we have a line segment and we can calculate its slope with the usual $\Delta y/\Delta x$:

$$\text{derivative of f at x} = \frac{f(x+h) - f(x)}{(x+h) - x} = \frac{f(x+h) - f(x)}{h}$$

We use the notation

$$\frac{df(x)}{dx}$$

for the slope (derivative) of a function f at point x.

The place where limits come in is that if we make h really small then this intersecting line will become the tangent line, and its slope will be the derivative at that point.

Note: The notation $\frac{df(x)}{dx}$ is called the "Leibniz notation". Other common ways of writing the derivative are the "Lagrange notation:" $f'(x)$, and Newton's "dot" notation which is usually used when we have functions of *time*:

$$\dot{x} = \frac{dx}{dt}$$

15.10 Visualizing derivatives with an animation

Drawing the pictures for the derivative on the board might help some, but an animation really drives home the idea of the derivative as a *limit*.

The program in Listing **??** shows an animation of the types of plots from a *secant* line to *tangent* line. See Figure **??** and Figure **??**.

Listing 15.10.1: derivative-animation.py - look at the limit as h approaches zero: the secant line becomes a tangent line at the given point.

```
#! /usr/bin/env python3

import math
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

draw_scale = 4.0

def main():
    # the limits and the function that we plot
    a = -1.3
    b = 3.2
    func = curvy_cubic

    # matplotlib graphics setup - boiler-plate stuff
    plt.rcParams.update({'font.size': 10*draw_scale})
    fig = plt.figure(figsize=(6.4*draw_scale, 4.8*draw_scale))

    # first generate the x for the entire domain from a to be
    x = np.arange(a, b, 1.0/1000)
    # pick the point at which we calculate the derivative
    # x0 = a + 2.0*(b-a)/3
    x0 = 1.8

    ## now a loop in which h gets smaller and smaller
    for i in range(2,300):
        # pick a value of h that keeps decreasing toward 0
        h = 2/(i + 1e-2*i*i)
        # calculate the slope of the secant line
        y0 = func(x0)
        y1 = func(x0+h)
        m = (y1 - y0) / h
        # now get the intercept
        b = y0 - m * x0
        print(x0, h, x0+h, y0, m, b)

        # to animate we clear and then redraw everything
        fig.clear()
        # draw the entire curve
        plt.plot(x, func(x), color='g', linewidth=1.0*draw_scale)
        plt.grid(True)
        # draw the two secant points
        plt.plot([x0], [y0], marker='o', markersize=7*draw_scale, color='red')
        plt.plot([x0+h], [y1], marker='o', markersize=5*draw_scale, color='black')
        # draw the line through those two points, using the linear
        # equation y = m*x + b
        line_x_pts = np.arange(x0 - 1.2, x0 + 1.2, 1.0/100)
        line_y_pts = m * line_x_pts + b
        plt.plot(line_x_pts, line_y_pts, color='b', linewidth=1.0*draw_scale)
```

(continues on next page)

(continued from previous page)

```
        # annotate the current value of h and slop
        info_para = """func: %s\nh: %g\nslope: %g""" % (func.__name__, h, m)
        plt.annotate(
            info_para,
            xy=(0, -2), xytext=(0, -2))
        # now annotate where (x0, y0) and (x0+h, y1) are
        plt.annotate('(x0, y0)', xy=(x0, y0), xytext=(x0-1, y0-3),
                     arrowprops=dict(facecolor='black', shrink=0.05))
        plt.annotate('(x0+h, y1)', xy=(x0+h, y1), xytext=(x0+h+1, y1+2),
                     arrowprops=dict(facecolor='black', shrink=0.05))
        # to animate we do the draw_idle() and pause()
        fig.canvas.draw_idle()
        plt.pause(1.45+h)

    plt.waitforbuttonpress()

def curvy_cubic(x):
    """a simple cubic function mixed with a sin - it shows some
    interesting curves"""
    return (x-1)**3 - 3*(x-1) + np.sin(x)

main()
```

CHAPTER

SIXTEEN

NUMERICAL INTEGRATION

[status: enough written to run a course, but incomplete]

16.1 The integral

The *integral* of a function is the "area under the curve" of that function between two given points a and b. We write it in this way:

$$\int_a^b f(x)dx \tag{16.1.1}$$

A picture that illustrates this is in the Wikipedia article on integrals which I reprodue here in Figure **??**.

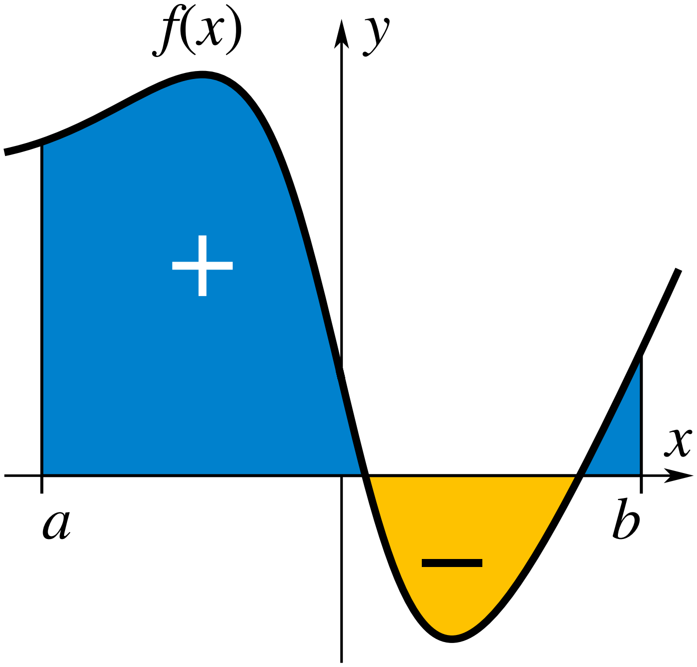


Figure 16.1.1: The integral is the area under the curve of a function, between two points a and b.

Other written tutorials explain integrals quite well, but the figure above allows an instructor to explain it at a blackboard or whiteboard.

For now remember a couple of things: the integral is an area, and notice the beautiful notation in (**??**).

Integrals come up a lot in science. In Physics, to example, *work* is defined as the integral of force over a distance. This way of describing of work then leads to defining a body's *energy*. *Electric flux* is the integral of the electric field over a surface (a 2-dimensional integral, but the same idea). A simple application in pure math is to calculate the areas of certain figures and the volumes of some solids.

Integrals are related to derivatives by the *fundamental theorem of calculus*, developed in the 17th century and based on work by Archimedes in ancient Greece. I will not touch on this too much at this time: it's possible to explain derivatives and integrals to kids, but explaining the "antiderivative" might be out of our reach. We will just occasionally mention little bits of information about the link.

16.2 Calculating the integral numerically

It rare that we can calculate an integral exactly, but we can write computer programs that approximate the integral of any function. In Figure **??** we see one way of doing it by breaking the area down into many small rectangles. The area is then close to the sum of the areas of those small rectangles.

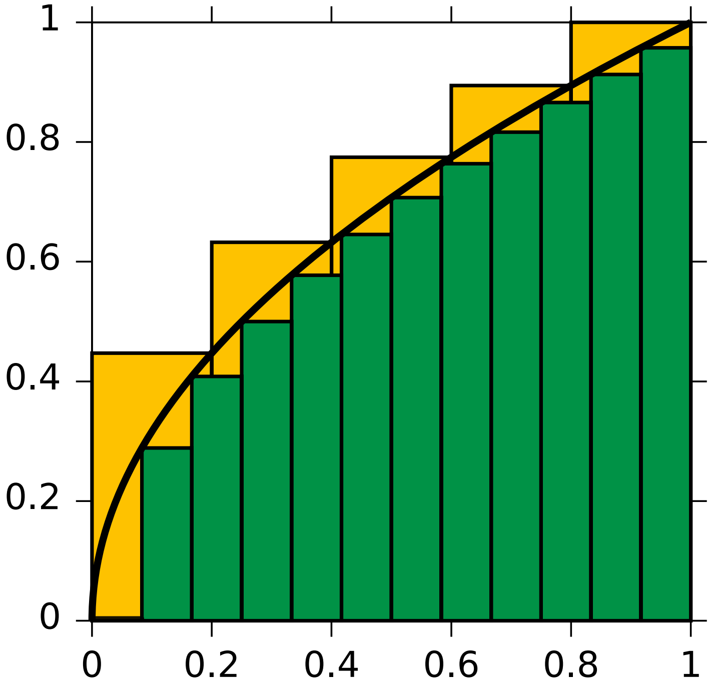


Figure 16.2.1: Approximating the area under the curve by making many small rectangles that try to fit the curve.

Figure **??** shows how we can break that region of space into a bunch of rectangles. The area of all those rectangles will approximate the integral of our function. If you insert more rectangles, they will snuggle up to the curve even better.

The drawing in Figure **??** lets us explain the notation in Equation (**??**). Each little rectangle in the figure has a base of dx, and a height of $f(x)$. The area of that little rectangle is height × base, which is $f(x)dx$. Now think of $\int$ as a fancily drawn letter S and you see that the notation in Equation (**??**) reads as "the sum of $f(x)dx$ for x between a and b", which means "the sum of the areas of all the baby rectangles between a and b".

The key insight is then to notice that in the *limit* (yes! it's limits again!) of an increasing number of smaller and smaller rectangles we get the the integral.

Let us write a program which calculates the area under a curve using this approximation. Students are encouraged to think for a while before looking at Listing **??**. I spend some time devising an algorithm with the students. I typically go to the whiteboard and, interactively with the students, I point out that we need:

- A loop over the number of rectangles.
- A variable with the area we have accumulated so far. (This is a standard paradigm in many programs: a variable that starts at zero, and you add stuff to it as you compute it.)
- Identifying the lower left and right corners of each baby rectangle.
- Finding the upper left and right corners of each baby rectangle.
- Finding the height of the rectangle using $f(x)$.
- Taking that individual rectangle's area.
- Adding that rectangle's area to the sum variable.

Now let us write the program in Listing **??** program and run it. It will report the area it found. If you put few rectangles you will get a worse approximation, and if you put in more rectangles you get a better approximation.

Listing 16.2.1: integrate-with-rectangles.py - approximate the area under a curve with a collection of small rectangles that fit under that curve.

```
#! /usr/bin/env python3

import math

def main():
    a = -1
    b = 1
    n_rectangles = 100

    # A = integrate_function(curvy_cubic, a, b, n_rectangles)
    A = integrate_function(upper_semicircle, a, b, n_rectangles)
    print('with %d rectangles, the area is %g' % (n_rectangles, A))

def integrate_function(f, a, b, n_rectangles):
    width = (b - a) / n_rectangles
    total_area = 0
    for i in range(n_rectangles):
        midpoint = a + i * width + width/2
        height = f(midpoint)
        area = width * height
        total_area += area
    return total_area

def curvy_cubic(x):
    return (x-1)**3 - 3*(x-1) + math.sin(x)

# Note that this function, when integrated from -1 to 1, should give
# an area of pi/2
def upper_semicircle(x):
    return math.sqrt(1 - x*x)

main()
```

You can modify integrate-with-rectangles.py to use a curvy function curvy_cubic() or to use the upper_semicircle() function. The latter is interesting because the integral under the upper semicircle should be half the area of a circle with that radius: πr^2. In our case the radius is 1, so we expect that $\int_{-1}^{1} \mathrm{upper\_semicircle}(x) dx = \pi/2$. This means that we can use our integration program to calculate π!

But integrate-with-rectangles.py was a rather dry program: it is simple to write and understand, it gives the answer, but it does not carry us on a wave of enthusiasm.

So let us see if we can make an animation of the process of this approximation. Enter the program in Listing **??**. This version shows the plot of the curve as well as the subdivision of the area into rectangles, and it gives us an update on the estimate of the area.

Listing 16.2.2: integrate-with-rectangles-and-plot.py - calculate area with the same approximation in Listing **??** but also draw an animation of the rectangles as we do it.

```
#! /usr/bin/env python3

import math
import matplotlib.pyplot as plt
import numpy as np

def main():
    a = -1.3
    b = 3.1
    func = curvy_cubic
    # a = -1
    # b = 1
    # func = upper_semicircle

    fig = plt.figure()
    for n_rectangles in range(5, 200, 3):
        do_single_integration(fig, func, a, b, n_rectangles)
    plt.waitforbuttonpress()

def do_single_integration(fig, func, a, b, n_rectangles):
    x = np.arange(a, b, 1.0/n_rectangles)
    fig.clear()
    plt.grid(True)
    plt.plot(x, func(x), color='g')
    A = integrate_function(func, a, b, n_rectangles)
    info_para = """func: %s
n_rectangles: %d
area: %g
""" % (func.__name__, n_rectangles, A)
    plt.annotate(
        info_para,
        xy=(0, -1), xytext=(0, -2))
    print(info_para)
    fig.canvas.draw_idle()
    plt.pause(0.8)

def integrate_function(f, a, b, n_rectangles):
    width = (b - a) / n_rectangles
```

(continues on next page)

(continued from previous page)

```
    total_area = 0
    for i in range(n_rectangles):
        left_point = a + i*width
        right_point = a + (i+1)*width
        midpoint = a + i * width + width/2
        height = f(midpoint)
        area = width * height
        total_area += area
        # now plot that rectangle; we must renormalize all to be in
        # the [0,1] range for both x and y
        if f(midpoint) < 0:
            color = 'r'
        else:
            color = 'b'
        norm_left = left_point
        plt.bar([midpoint], height, width=width, edgecolor=color, fill=False)
    return total_area

def curvy_cubic(x):
    return (x-1)**3 - 3*(x-1) + np.sin(x)

# Note that this function, when integrated from -1 to 1, should give
# an area of pi/2
def upper_semicircle(x):
    return np.sqrt(1 - x*x)

main()
```

Run the program and pay close attention to the ongoing calculation of the area and see if it converges as you get more and more rectangles.

16.3 Improving the numerical approximation

If you imagine using *trapezoids* instead of rectangles in Figure **??** you will notice that they hug the curve more closely. The area of a trapezoid is straightforward to calculate (I usually get the students to work it out while I write their ideas on the whiteboard). This method is called the *trapezoidal rule*.

Exercise 16.1: trapezoidal rule Modify the program in Listing **??** to use *trapezoids* instead of rectangles. Compare the results to those from the rectangular method.

Even better is a method called *Simpson's rule*. We take three points along the line and approximate the curve between them with a second degree polynomial. The curve then becomes a stitched together sequence of baby parabolas. Parabolas hug our function curve even more closely, so we should get a better approximation.

The procedure for Simpson's rule has three main steps:

1. Break the interval from a to b into segments which have a left, middle and right point.
2. Find the parabolas using techniques similar to those in Section **??**.
3. We know how to integrate polynomials exactly, so each tiny parabola has a known area, for example $\int_a^b x^2 dx = (b^3 - a^3)/3$.

Exercise 16.2: Simpson's rule Modify the program in Listing **??** to use Simpson's rule.

16.4 Stepping back from numerical integration back to analytical work

CHAPTER

SEVENTEEN

DIFFERENTIAL EQUATIONS

Section author: Sophia Mulholland <smulholland505gmail.com>

17.1 Motivation, Prerequisites, Plan

The goal of this chapter is to learn about differential equations and how they are used to solve scientific problems. We will also learn to write programs in C to solve differential equations.

Differential equations are used to answer scientific questions about how systems *change*. We will look in to these examples:

A very loose definition of the uses of differential equations is to describe how things change.

- how springs move
- how populations grow
- how radioactive material decays
- how bridges collapse

The plan is to start out with a simple explanation of derivatives, and see examples of derivatives in action. We'll look into the derivation of a falling object's equation and then Euler's method. Then we'll end up modeling a falling body and an oscillating spring.

17.2 Derivatives

If you google the word 'derivative' you'll find the definition:

An expression representing the rate of change of a function with respect to an independent variable.

So the rate of change of a function can change right? It could be going up then down and then up again. So how can we represent the rate of change if it's not just a number? We use an expression. Like $2x$ or $5x^3$. We can use a function to model the rate of change of another function.

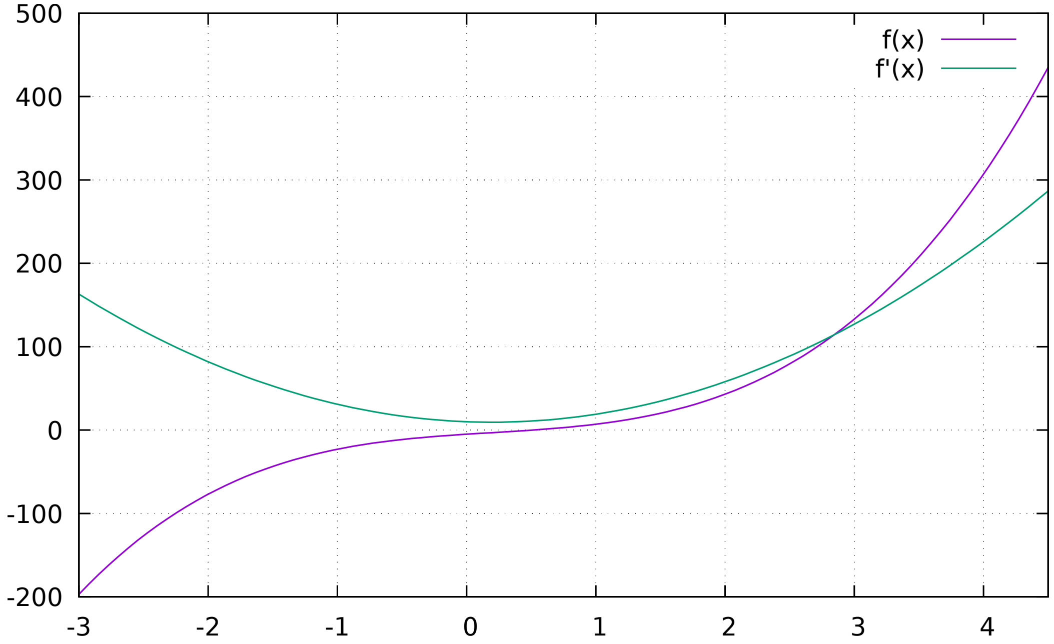
Here's an example of taking a derivative:

$$f(x) = 5x^3 - 3x^2 + 10x - 5$$
$$f'(x) = 15x^2 - 6x + 10$$

There are many different techniques to solve these equations like the Chain rule, the Quotient rule, and the Product rule.

You can graph the functions in gnuplot like this

```
gnuplot
# and at the gnuplot> prompt:
set grid
plot 5*x*x*x-3*x*x+10*x-5
plot 15*x*x-6*x+10
```



17.2.1 Why?

So why do we care about the rate of change of a function? It helps us predict what's going to come next. We can predict the future population of earth by analyzing its rate of change right now and making an estimate. We can predict how radioactive material is going to decay.

You can think of a derivative as an instantaneous rate of change.

In the image below, as the two blue points get closer together, we can estimate the slope as the change in x as it approaches 0.

There are real world examples of derivatives all throughout physics, computer science, chemistry, biology, and economics.

Derivatives are an instantaneous rate of change or a slope at one point in time.

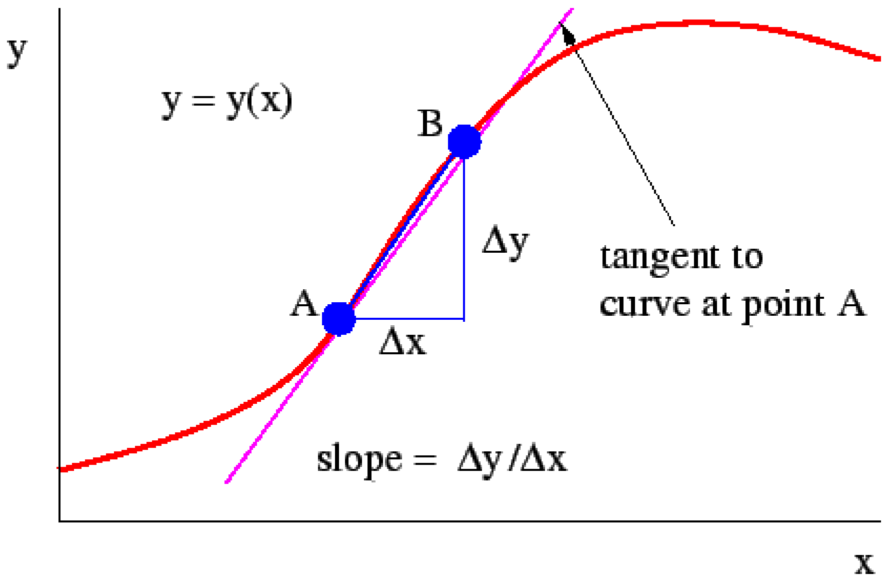


Figure 17.2.1: This photo from the Wikipedia article on derivatives, https://en.wikibooks.org/wiki/Waves/Derivatives shows the concept of taking the derivative

17.3 Definitions

Differential equation

An equation involving a function and one or more of it's derivatives

Derivative

An expression representing the rate of change of a function with respect to an independent variable.

Different notations:

$$\begin{aligned} y'' &= 3x \text{ Lagrange notation} \\ f''(x) &= 3x \text{ Lagrange notation} \\ \frac{d^2y}{dx^2} &= 3x \text{ Leibniz notation} \\ \ddot{y}(t) &= 3t \text{ Newton notation (only for functions of time)} \end{aligned}$$

We have heard lots of talk of equations, and it was probably a reference to *algebraic* equations, which might look like:

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

We work out the solutions, possibly by factoring it:

$$\begin{aligned} (x+3)(x-1) = 0 \\ \cdots \implies x = \{1, -3\} \end{aligned}$$

The solution to a system of algebraic equations is a number or set of numbers.

But now we talk about *differential* equations. An example might look like (in the various notations):

$$\begin{aligned} y'' + 2y' = 3y \\ f''(x) + 2f'(x) = 3f(x) \\ \frac{d^2y}{dx^2} + 2\frac{dy}{dx} = 3y \\ \cdots \implies y = e^{-3x} \end{aligned}$$

The solution to a differential equation is a function or a family of functions.

17.4 An Example

We know from Newton's law that

$$F = ma$$

If the force is the force gravity for a falling body, $F = -mg$, then:

$$ma = -mg$$

and that

$$a = -g$$

Now

$$\begin{aligned} ma &= -mg \\ m\frac{d^2y}{dt^2} &= -mg \\ \frac{d^2y}{dt^2} + g &= 0 \end{aligned}$$

From this, we have proved that the constant $-g$ is equal to the second derivative of position.

We also know that acceleration is the second derivative of position. How? Well if an object's position is x, then it's velocity is the change in x over time.

The acceleration is then the change in velocity over time. $\frac{dv}{dt}$ or $\frac{d^2x}{dt^2}$

17.5 Population Growth

The differential equation describing population growth is:

$$\frac{dP(t)}{dt} = rP(t)$$

One way to read this is to think that the more rabbits you have, the more babies you will make in a generation.

How do you solve this? Do you remember which function had a derivative that was equal to itself?

$$\begin{aligned} f(x) &= e^x \\ \frac{df(x)}{dx} &= e^x = f(x) \end{aligned}$$

with a constant multiplier in the exponent:

$$\begin{aligned} f(x) &= e^{\alpha x} \\ \frac{df(x)}{dx} &= \alpha f(x) \end{aligned}$$

With all that in hand, we can *guess* that the solution is: $P(t) = e^{rt}$: this will solve the differential equation.

But wait, now note that if you multiply $P(t)$ by a constant A, the equation is still satisfied, since you can simplify away any constant multiple of $P(t)$ in the equation. This means that the most general solution is:

$$P(t) = A \times e^{rt}$$

While a pure mathematician might be happy to simply say that you can have an arbitrary constant, a scientist will naturaly ask what the constant means in the system we are studying. And usually there is a good answer. Take a look at what happens at time $t = 0$:

$$P(0) = Ae^{r \times 0} = Ae^0 = A$$

So A is the starting value of the population at time 0. We can call it P_0 and write:

$$P(t) = P_0 e^{rt}$$

To summarize our notation in this problem:

- $P(t)$ is the population at time t
- P_0 is the initial population
- and r is the growth rate

Let's try to graph a rabbit population with an initial population of 2 rabbits and a growth rate of 0.5:

$P = 2e^{0.5t}$ is the function

$\frac{dP}{dt} = e^{0.5t}$ is the function's derivative

```
gnuplot
# and at the gnuplot> prompt:
plot 2*exp(0.5*x)
replot exp(0.5*x)
```

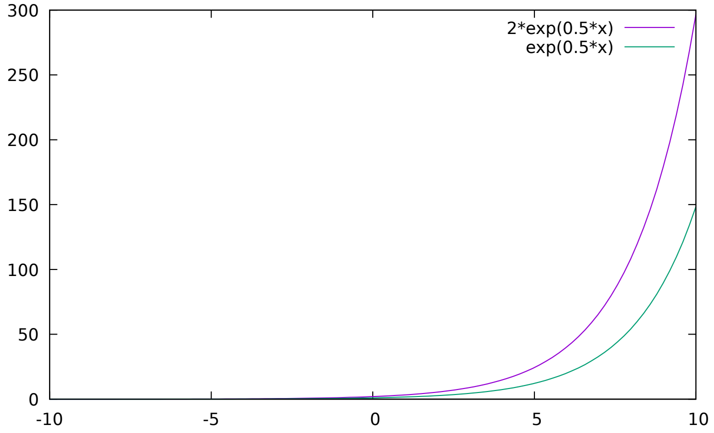


Figure 17.5.1: The growth of the population is in purple and the derivative of that line is in green. Note that as the slope of the rabbit population gets steeper, the derivative also increases.

17.5.1 But what does the derivative tell us?

- Since the derivative is positive, the function is increasing.
- Since the derivative is increasing, the slope of the function is increasing.

17.6 Euler's Method

Euler's Method is a way of approximating the solution of a first-order differential equation. Remember that the solution of a differential equation is always a function or set of functions. To use this method, we need to know the first point on the section we want to look at.

When we don't know the solution to a particular equation, we can use Euler's Method. By taking little steps along the graph, it shows us a prediction of what the exact solution looks like.

Figure 17.6.1: From Wikipedia, https://en.wikipedia.org/wiki/Euler_method A picture of the Euler method. The unknown curve is blue, and the approximation is in red.

The idea is that if you have the differential equation and one starting point, you can approximate the solution of the equation. In the figure above, the red line is obviously not curvy like the blue, but why?

At the first point, we have the derivative at a point (x_1, y_1). That means we have the slope at that exact point. We use the slope to graph a tangent line across a time step, which is why they appear to be segments. Then, we have the next two points (hopefully they are close to the actual solution). We calculate the derivative for (x_2, y_2) and repeat the process.

Let's graph $y' = 2x - 2$ and use euler's method to graph its solution $y = x^2 - 2x$.

It's much easier to know if we're right, if we already know the solution. For most differential equations, it is really hard or impossible to solve it analytically.

Listing 17.6.1: `euler-method.c` demonstrates Euler's method.

```
// this program uses Euler's method to approximate solutions to the
// differential equation dy/dx = f(x, y)

// compile with: gcc -o euler-method euler-method.c -lm
// run with: ./euler-method > euler-method.dat
// plot in gnuplot with:
// plot "euler-method.dat" using 1:2 with lines
// replot  "euler-method.dat" using 1:3 with lines

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* prototypes for functions below */
double RHS_parab(double x, double y);
double exact_parab(double x, double y);


int main(int argc, char *argv[])
{
  double xinitial = -5;
```

(continues on next page)

(continued from previous page)

```
  double yinitial = 35;
  double interval = 10;         /* how many seconds */

  /* handle command line options that allow the user to select number
     of steps */
  int n_steps;
  if (argc == 1) {
    n_steps = 1000;
  } else if (argc == 2) {
    n_steps = atoi(argv[1]);
  } else {
    fprintf(stderr, "error: usage is %s [n_steps]\n", argv[0]);
    return 1;
  }

  /* put out a wee bit of metadata with the "low effort metadata"
     (LEM) structured comment approach */
  printf("##COMMENT: Euler method solution to dy/dx = f(x, y)\n");
  printf("##N_STEPS: %d\n", n_steps);
  printf("##COLUMN_DESCRIPTIONS: x    y_approx    y_exact   slope\n");

  double dx = interval/n_steps;
  double xprev = xinitial;
  double yprev = yinitial;
  double exact = exact_parab(xprev, yprev);
  printf("%lf   %lf   %lf\n", xprev, yprev, exact);
  for (int j = 0; j < n_steps; j++) {
    // calculate right hand side (RHS) with (xi, yi)
    double slope = RHS_parab(xprev, yprev);
    // increase x by step size
    double xnew = xprev + dx;
    // use tangent line to approximate next y value from previous y value
    double ynew = yprev + dx * slope;
    // find exact exact_solution to compare approximation with
    double exact = exact_parab(xnew, ynew);
    printf("%g   %g   %g   %g\n", xnew, ynew, exact, slope);
    // now reset the cycle
    xprev = xnew;
    yprev = ynew;
  }
  return 0;
}

// returns RHS (right hand side)
double RHS_parab(double x, double y)
{
  return 2*x - 2;
}

// returns exact exact_solution
double exact_parab(double x, double y)
{
```

(continues on next page)

(continued from previous page)

```
  return x*x - 2*x;
}
```

Compile and run and plot this with:

```
gcc -o euler-method euler-method.c -lm
./euler-method > euler-method.dat
gnuplot
# then at the gnuplot> prompt type:
set grid
plot "euler-method.dat" using 1:2 with lines
replot  "euler-method.dat" using 1:3 with lines
```

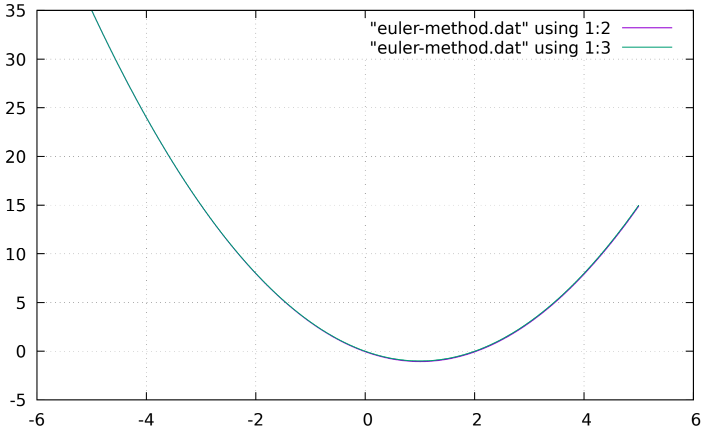


Figure 17.6.2: Euler's approximation with 1000 steps.

At first glance, It looks like the two lines are right on top of each other right? Well if you zoom in on your own graph you can probably see that the lines are not exactly the same. The green line is our exact solution. Using Euler's method can give us a good approximation for the solution, but never exact.

So now, we can compare that good approximation to a poor approximation using Euler's method. If you run:

```
./euler-method 10 > euler-method-bad-approximation.dat
gnuplot
# then at the gnuplot> prompt type:
set grid
plot "euler-method-bad-approximation.dat" using 1:2 with lines title "approximate"
replot  "euler-method-bad-approximation.dat" using 1:3 with linespoints title "exact"
```

On the purple line there are only ten steps and so it differs a lot from an exact solution of the equation $\frac{dy}{dx} = 2x - 2$.

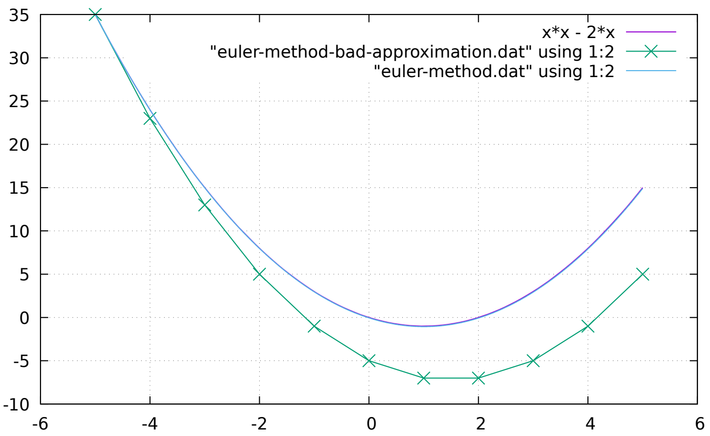


Figure 17.6.3: Euler's approximation with just 10 steps.

17.7 Second Order Differential equations

A second order differential equation is one that involves not just the first derivative of y, but the second one too.

$$y'' + P(x)y' + Q(x)y = 0$$

where $P(x)$ and $Q(x)$ are continuous functions or constants.

The solutions to second order differential equations will have two arbitrary constants, instead of having just one for first order equations. Yes: this is not a coincidence.

Incorporating a second order differential into a real world problem can be really useful. For instance, a model of a ball dropped off a tall building is going to have more than one factor right? To model it's position, you have to incorporate it's initial position AND its initial velocity AND it's acceleration due to gravity.

17.8 Falling Body

Imagine taking a basketball and dropping it off a building. What force is acting on that ball?

Gravity will cause the ball to accelerate towards the ground. So what would we need to model it? The mass of the basketball and the acceleration of gravity.

The way this is described in physics is with *Newton's second law:*

$$F = ma = m\frac{d^2y}{dt^2}$$

We know that for a smallish fall in an ordinary part of the world the force of gravity is $F = -mg$ (the negative sign is because it is going down), so we end up with this equation:

$$m\frac{d^2y}{dt^2} = -mg$$

you can cancel the mass terms to get:

$$\frac{d^2y}{dt^2} = -g$$

The solution to this equation is one of the simplest cases of indefinite integral applied twice:

$$v(t) = -gt + C_1$$
$$y(t) = -\frac{1}{2}gt^2 + C_1 t + C_2$$

where $C1$ and C_2 are the two arbitrary constants we expect from a second order differential equation. Note that at time $t = 0$ we can see that

$$v(0) = C_1$$
$$y(0) = C_2$$

so we can pick better names for the constants: v_0 and y_0 (initial velocity and initial height):

$$v(t) = gt + v_0$$
$$y(t) = -\frac{1}{2}gt^2 + v_0 t + y_0$$

where the physical meaning of the arbitrary constants stands out nicely.

Type in the program below called `falling-body.c`

Listing 17.8.1: `falling-body.c` shows exact solution and approximation

```c
// this program uses Euler's method to approximate solutions to the
// falling body equation d^2y/dt^2 = -g

// compile with: gcc -o falling-body falling-body.c -lm
// run with: ./falling-body > falling-body.dat
// plot in gnuplot with:
// plot "falling-body.dat" using 1:2 with lines
// replot  "falling-body.dat" using 1:3 with lines

#include <stdio.h>
#include <math.h>

//acceleration due to gravity
double g = 9.8;
double yinit = 100;
double vinit = 0.1;

double acceleration(double t, double y)
{
  return -g;
}

// returns exact solution
double falling_body_exact(double st)
{
  return yinit + vinit*st - 0.5*g*pow(st,2);
}

int main()
{
  double tinit = 0;
```

(continues on next page)

(continued from previous page)

```

  // define initial variables, step size
  double duration = 10; /* how many seconds */
  int n_steps = 10000;
  double dt = duration/n_steps;

  printf("##COMMENT: Euler method solution to falling body d^y/dx^2 = -g\n");
  printf("##N_STEPS: %d\n", n_steps);
  printf("##COLUMN_DESCRIPTIONS: t    y_approx   v_approx    y_exact   acc\n");

  // set the variables based on our initial values
  double tprev = tinit;
  double yprev = yinit;
  double vprev = vinit;
  double exact = falling_body_exact(tprev);
  double acc = acceleration(tprev, yprev);
  printf("%g   %g   %g   %g   %g\n", tprev, yprev, vprev, exact, acc);
  for (int j = 0; j < n_steps; j++) {
    // solve differential with initial values
    acc = acceleration(tprev, yprev);
    // using acceleration, update velocity
    double vnew = vprev + dt * acc;
    // now, using velocity, update position
    double ynew = yprev + vnew * dt;
    // increase t by step size
    double tnew = tprev + dt;
    tprev = tnew;
    yprev = ynew;
    vprev = vnew;
    // find exact solution to compare approximation with
    double exact = falling_body_exact(tnew);
    printf("%g   %g   %g   %g   %g\n", tnew, ynew, vnew, exact, acc);
  }
  return 0;
}
```

Compile and run it with:

```
gcc -o falling-body falling-body.c -lm
./falling-body > falling-body.txt
```

Now use gnuplot to graph the falling body as a function of time

```
gnuplot
# then at the gnuplot> prompt:
set grid
plot "falling-body.txt" using 1:2 with lines
replot "falling-body.txt" using 1:4 with lines
```

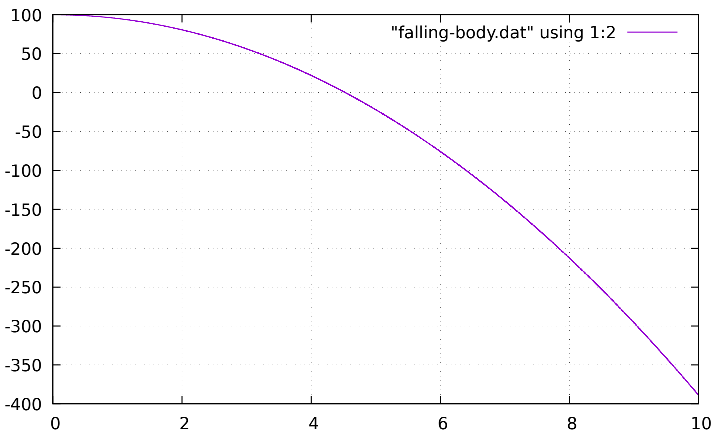


Figure 17.8.1: Height of a falling body versus time.

17.8.1 Adding Air Resistance to Falling Body

If you consider air drag then the falling body behaves differently. Air drag can often be modeled with an opposing force

$$F_{\text{drag}} = bv^2 = b\dot{y}^2$$

where b is a constant coefficient which depends on the properties of the fluid (in our case air) and the body moving in it.

This drag force gives us this differential equation:

$$m\ddot{y} = b\dot{y}^2 - mg$$
$$m\ddot{y} - b\dot{y}^2 + mg = 0$$

The full solution to this differential equation is possible, though quite gnarly - we discuss the full solution below.

Type or paste in the program below called `falling-body-drag.c`

Listing 17.8.2: `falling-body-drag.c` shows a falling body's path as a function of time.

```
// this program uses Euler's method to approximate solutions to the
// falling body equation with air drag: d^2y/dt^2 + g

// compile with: gcc -o falling-body-drag falling-body-drag.c -lm
// run with: ./falling-body-drag > falling-body-drag.dat
// plot in gnuplot with:
// plot "falling-body-drag.dat" using 1:2 with lines
// replot  "falling-body-drag.dat" using 1:3 with lines

#include <stdio.h>
#include <math.h>

//acceleration due to gravity
double mass = 1;                /* kg */
double g = 9.8;                 /* m/s^2 */
double yinit = 100;             /* m */
```

(continues on next page)

(continued from previous page)

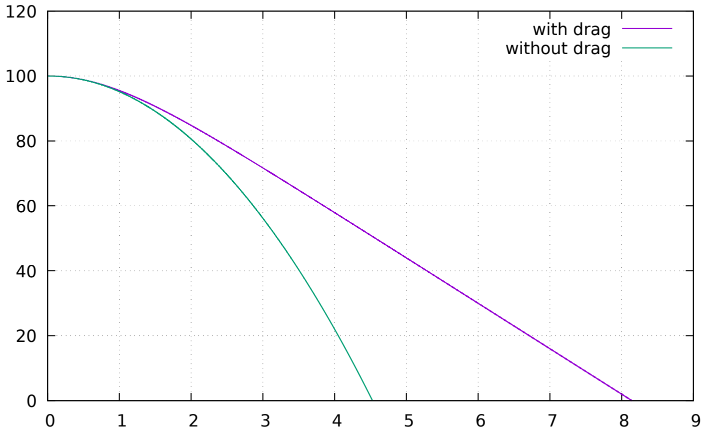
```
double vinit = 0.1;             /* m/s */
double b_drag = 0.05;            /*  */

double force(double t, double y, double v)
{
  return -mass*g + b_drag * v * v;
}

// returns exact solution
double falling_body_exact(double st)
{
  return yinit + vinit*st - 0.5*g*pow(st,2);
}

int main()
{
  double tinit = 0;

  // define initial variables, step size
  double duration = 9; /* how many seconds */
  int n_steps = 10000;
  double dt = duration/n_steps;

  printf("##COMMENT: Euler method solution to falling body d^y/dx^2 = -g\n");
  printf("##N_STEPS: %d\n", n_steps);
  printf("##COLUMN_DESCRIPTIONS: t    y_approx   v_approx no_drag_exact acc\n");

  // set the variables based on our initial values
  double tprev = tinit;
  double yprev = yinit;
  double vprev = vinit;
  double exact_nodrag = falling_body_exact(tprev);
  double acc = force(tprev, yprev, vprev) / mass;
  printf("%g   %g   %g   %g   %g\n", tprev, yprev, vprev, exact_nodrag, acc);
  for (int j = 0; j < n_steps; j++) {
    // solve differential with initial values
    acc = force(tprev, yprev, vprev) / mass;
    // using acceleration, update velocity
    double vnew = vprev + dt * acc;
    // now, using velocity, update position
    double ynew = yprev + vnew * dt;
    // increase t by step size
    double tnew = tprev + dt;
    tprev = tnew;
    yprev = ynew;
    vprev = vnew;
    // find exact solution to compare approximation with
    double exact_nodrag = falling_body_exact(tnew);
    printf("%g   %g   %g   %g   %g\n", tnew, ynew, vnew, exact_nodrag, acc);
  }
  return 0;
}
```

Compile and run it with

```
gcc -o falling-body-drag falling-body-drag.c -lm
./falling-body-drag 1.7 1000 0.4 > falling-body-drag.dat
```

Now use gnuplot to graph the falling body as a function of time, including drag:

```
set grid
plot "falling-body-drag.dat" using 1:2 with lines title "with drag"
replot "falling-body-drag.dat" using 1:4 with lines title "without drag"
```



You will notice that the two plots start out together, but then the plot with drag diverges from the free fall and eventually straightens out.

This straightening out of the slope of $y(t)$ means that the velocity has become a constant due to air drag balancing out the effects of gravity. This is called the *terminal velocity*.

To get more insight into what happened you can look at the plots of column 3 (velocity) and column 5 (acceleration). You will see that the velocity starts close to 0, gets very negative very quickly, and then tapers off to about $-14m/s$, which is the terminal velocity.

The acceleration, instead, starts at $-9.8m/s^2$ and taperes off to zero once the body is moving fast enough that the bv^2 drag force matches the force of gravity.

Now for a discussion of the full solution. There is an article at: https://philosophicalmath.wordpress.com/2017/10/21/terminal-velocity-derivation/ in which they focus on the question of calculating the velocity.

This they find to be:

$$arctanhv(t) = \sqrt{\frac{mg}{b}} \tanh\left(t\sqrt{\frac{bg}{m}} + \left(v_0\sqrt{\frac{b}{mg}}\right)\right)$$

The $\tanh()$ function is integrable, so one can also find $y(t)$ exactly from this equation, but their main focus is on how to find the terminal velocity:

$$\lim_{t\to\infty} v(t) = \lim_{t\to\infty} (\dots) = \sqrt{\frac{mg}{b}}$$

So the terminal velocity depends on the mass, the acceleration of gravity, and (inversely) on the coefficient b. This matches our intuition, and if we look at the values in our program (at the time of writing this paragraph):

$$\begin{aligned} m &= 1\ kg \\ g &= 9.8\ m/s^2 \\ b &= 0.05\ \text{kg}/m \\ \ldots \implies v_{\text{terminal}} &= 14\ m/s \end{aligned}$$

which is what our numerical solution found.

One more thing to say about drag: when certain conditions are satisfied for an object falling in a fluid (or having the fluid move around it), the coefficient b can be given by:

$$b = \frac{1}{2}\rho C_d A$$

where ρ is the fluid density, C_d is the drag coefficient, and A is the cross-sectional area of our falling body.

17.9 The harmonic oscillator

The motion of a mass attached to a spring is described by Newton's law, where the force produced by the spring is given (once you ignore all friction effects) by Hooke's law:

$$F = -kx$$

Here is how to read that equation: x is a small displacement of the mass when you pull it from its rest position. k is the *sprint constant* - a bigger value of k means that we have a stiffer spring which will return to its resting point more vigorously.

If you *push* the spring instead of pulling you get (for small displacements) the same force in the opposite direction.

The resulgint system is called a "harmonic oscillator". The Wikipedia page on the harmonic oscillator points out its importance in physics with (approximately) this wording:

> The harmonic oscillator model is very important in physics, because any mass subject to a force in stable equilibrium acts as a harmonic oscillator for small vibrations. Harmonic oscillators occur widely in nature and are exploited in many human-made devices, such as clocks and radio circuits. They are the source of virtually all sinusoidal vibrations and waves.

17.9.1 The simple harmonic oscillator

Applying Newton's law, as we did for the falling body but this time with the spring force, we get:

$$\begin{aligned} m\frac{d^2x}{dt^2} &= -kx \\ \implies m\frac{d^2x}{dt^2} + kx &= 0 \end{aligned}$$

Let us look at this differential equation in its simplest form. The question that comes to mind is: "what function, when you take its derivative twice, gives you itself back again with a minus sign?"

The answer to that is either $\sin()$ or $\cos()$, or an exponential with an imaginary multiple of the argument.

So we expect the solution to look something like:

$$x(t) = A\cos(Ct) + B\sin(Ct)$$

where A and B are the familiar arbitrary constants, and C is some kind of folding of the parameters of the differential equation, like mass and spring constant.

A harmless simplification is to assume that we are plucking and letting go with an initial velocity of 0, which makes the $\sin()$ term go away, and we are left with just the cosine term:

$$x(t) = A\cos(Ct)$$

If we plug that into the differential equation we see that it works when we set the parameter C to be:

$$C = \sqrt{\frac{k}{m}}$$

We usually call this ω_0, the natural frequency of the oscillator:

$$\omega_0 = \sqrt{\frac{k}{m}}$$

and our solution is:

$$x(t) = A\cos(\omega_0 t)$$

This is a sinusoidal wave with amplitude A and frequency ω_0.

The velocity will be:

$$v(t) = -A\omega_0 \cos(\omega_0 t)$$

Think through the proportionalities involved here: the frequency is higher when the spring is stiffer, and lower when the mass is bigger. The same goes for the velocity. Does that match your intuition?

Without any other forces acting on it, this spring will keep going forever.

17.9.2 The damped harmonic oscillator

To model the spring as if it was happening in the real world, we need to account for friction. In physics the force given by sliding friction is usually proportional to the velocity:

$$F_{\text{friction}} = -C\frac{dx}{dt}$$

adding this to the spring force gives this differential equation for the damped harmonic oscillator:

$$m\frac{d^2x}{dt^2} + C\frac{dx}{dt} + kx = 0$$
$$\frac{d^2x}{dt^2} + \frac{C}{m}\frac{dx}{dt} + \omega_0^2 x = 0$$

This can be solved analytically and we get:

$$x(t) = Ae^{-\zeta\omega_0 t}\cos{(\omega t)}$$

If you look at this closely you see that:

- The frequency $omega$ is shifted a bit from the natural ω_0 of the undamped system. The full relationship is:

$$\omega = \omega_0\sqrt{1 - \left(\frac{C}{2\sqrt{mk}}\right)^2}$$

- There is a damping term $\exp(-\zeta\omega_0 t)$ where $\zeta = C/(2\sqrt{mk})$. This will make the oscillations damp out very quickly if C is big, and more gradually if C is small.

In the program `damped-spring-with-friction.c` below, you can model a damped oscillating spring.

Listing 17.9.1: `damped-spring-with-friction.c` shows exact solution and approximation for a spring.

```
// compile with: gcc -o damped-spring-with-friction damped-spring-with-friction.c -lm
// run with: ./damped-spring-with-friction > damped-spring-with-friction.dat
// plot in gnuplot with:
// plot "damped-spring-with-friction.dat" using 1:3 with lines
// replot  "damped-spring-with-friction.dat" using 1:4 with lines

#include <stdio.h>
#include <math.h>

//acceleration due to gravity
double g = 9.8;
//some physics values for the spring equation
double k = 3.0;
double m = 2.0;
//physics for air friction
double air_friction = 3;
//physics for frictional damping force
double damp = 0.5;
/* double damp = 0.0000005; */
double F0 = 10;                        /* constant driving force */
double amplitude = 2;

//returns differential
double acceleration(double t, double x, double v)
{
  double omega_0 = sqrt(k / m);
  return - damp * v / m - omega_0*omega_0 * x;
  /* return (-k*x - damp*v + F0*cos(8*t)) / m; */
}

// returns exact solution for harmonic motion
double harmonic_exact(double t)
{
  return amplitude*cos(sqrt(k/m - damp*damp/(4*m*m))*t)*exp((-damp/(2*m)) * t);
}

int main()
{
  double tinit = 0;

  // define initial variables, step size
  double duration = 30; /* how many seconds */
  int n_steps = 10000;
  double dt = duration/n_steps;

  printf("##COMMENT: Euler method solution to damped harmonic oscillator\n");
```

(continues on next page)

(continued from previous page)

```
  printf("###COMMENT: m d^y/dx^2 + C dx/dt + k x = 0");
  printf("##N_STEPS: %d\n", n_steps);
  printf("##COLUMN_DESCRIPTIONS: t    x_approx   v_approx no_damp_exact acc\n");

  // set the variables based on our initial values
  double tprev = tinit;
  double xprev = amplitude;
  double vprev = 0;              /* pluck, then release from rest position */
  double exact_harmonic = harmonic_exact(tinit);
  double acc = acceleration(tprev, xprev, vprev);
  printf("%g   %g   %g   %g   %g\n", tprev, xprev, vprev, exact_harmonic, acc);
  for (int j = 0; j < n_steps; j++) {
    // solve differential with initial values
    acc = acceleration(tprev, xprev, vprev);
    // using acceleration, update velocity
    double vnew = vprev + dt * acc;
    // now, using velocity, update position
    double xnew = xprev + vnew * dt;
    // increase t by step size
    double tnew = tprev + dt;
    tprev = tnew;
    xprev = xnew;
    vprev = vnew;
    // find exact solution to compare approximation with
    double exact_nodamp = harmonic_exact(tnew);
    printf("%g   %g   %g   %g   %g\n", tnew, xnew, vnew, exact_nodamp, acc);
  }
}
```

Run this program with

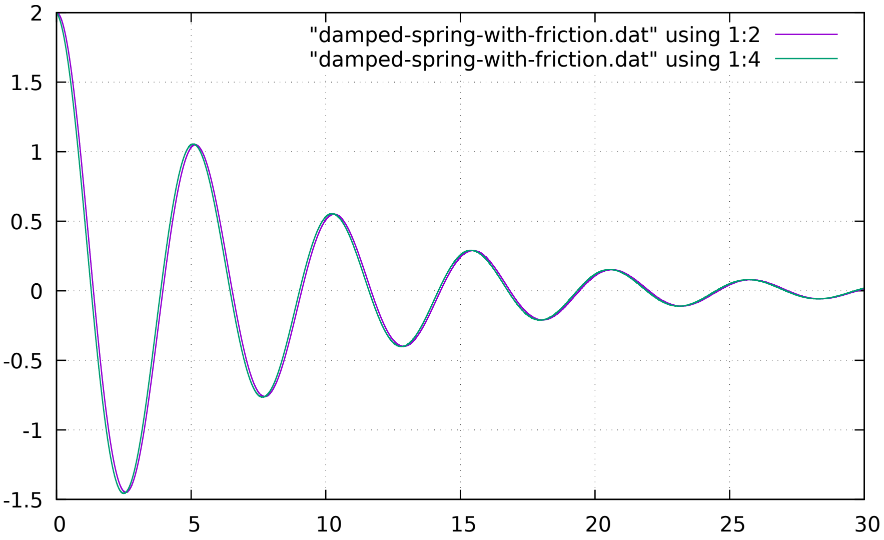
```
gcc -o damped-spring-with-friction damped-spring-with-friction.c -lm
./damped-spring-with-friction > damped-spring-with-friction.dat
```

Now let's use gnuplot to graph the damped spring:

```
gnuplot
# and at the gnuplot> prompt:
set grid
set title "damped harmonic oscillator"
set xlabel "time"
set ylabel "amplitude"
plot "damped-spring-with-friction.dat" using 1:2 with lines title  "damped oscillator␣
↪approximate"
replot  "damped-spring-with-friction.dat" using 1:4 with lines title "damped oscillator␣
↪exact"
```

It should look something like this:

If you look at the code you can see that you can do experiments by changing the `damp` parameter at the top, and you could even experiment wtih adding a driving force to the oscillator by changing the function `acceleration()`.



17.9.3 The non-linear pendulum

The force along the trajectory of mass on a string (pendulum) is:

$$F(t) = -mg\sin(\theta)$$

This gives us an expression of Newton's law as:

$$mL\frac{d^2\theta(t)}{dt^2} + mg\sin(\theta(t)) = 0$$
$$\frac{d^2\theta(t)}{dt^2} + (g/L)\sin(\theta(t)) = 0$$

For small angles we have $\sin(\theta) \approx \theta$, so we get:

$$\frac{d^2\theta}{dt^2} + (g/L)\theta(t) = 0$$

this last equation is the same as the simple harmonic oscillator equation, where our fundamental frequency is given by:

$$\omega_0 = \sqrt{\frac{g}{L}}$$

once again you should look at the proportionalities in that equation and see if you agree with them.

The full non-linear equation, where we have $\sin(\theta(t))$, is very difficult to solve. This has been solved in closed form, but the solutions are very complex and involve their own set of approximations to calculate *elliptical integrals.*

Since the solution is so complex, and the slightest variation in the kind of force we are contemplating would make the equation completely unsolvable, it becomes interesting to use our differential equation solver to calculate it.

Type or paste in the program `nonlinear-pendulum.c`:

Listing 17.9.2: `nonlinear-pendulum.c` approximates the equation for a nonlinear pendulum.

```c
// compile with: gcc -o nonlinear-pendulum nonlinear-pendulum.c -lm
// run with: ./nonlinear-pendulum > nonlinear-pendulum.dat
// plot in gnuplot with:
// plot "nonlinear-pendulum.dat" using 1:3 with lines
// replot  "nonlinear-pendulum.dat" using 1:4 with lines

#include <stdio.h>
#include <math.h>

//acceleration due to gravity
double g = 9.8;
//some physics values for the spring equation
double k = 3.0;
double m = 2.0;
//physics for air friction
double air_friction = 3;
//physics for frictional damping force
double damp = 0.5;
/* double damp = 0.0000005; */
double F0 = 10;                        /* constant driving force */
double theta_0 = 0.3;           /* radians */
double Length = 0.5;            /* length in meters */

double acceleration(double t, double x, double v)
{
  return - (g / Length) * sin(x);
  /* return (-k*x - damp*v + F0*cos(8*t)) / m; */
}

// returns exact solution for harmonic motion
double harmonic_exact(double t)
{
  return theta_0*cos(sqrt(g / Length) * t);
}

int main()
{
  double tinit = 0;

  // define initial variables, step size
  double duration = 10; /* how many seconds */
  int n_steps = 10000;
  double dt = duration/n_steps;

  printf("##COMMENT: Euler method solution to damped harmonic oscillator\n");
  printf("###COMMENT: m d^y/dx^2 + C dx/dt + k x = 0");
  printf("##N_STEPS: %d\n", n_steps);
  printf("##COLUMN_DESCRIPTIONS: t   x_approx   v_approx no_damp_exact acc\n");

  // set the variables based on our initial values
```

(continues on next page)

(continued from previous page)

```
  double tprev = tinit;
  double xprev = theta_0;
  double vprev = 0;             /* pluck, then release from rest position */
  double exact_harmonic = harmonic_exact(tinit);
  double acc = acceleration(tprev, xprev, vprev);
  printf("%g   %g   %g   %g   %g\n", tprev, xprev, vprev, exact_harmonic, acc);
  for (int j = 0; j < n_steps; j++) {
    // solve differential with initial values
    acc = acceleration(tprev, xprev, vprev);
    // using acceleration, update velocity
    double vnew = vprev + dt * acc;
    // now, using velocity, update position
    double xnew = xprev + vnew * dt;
    // increase t by step size
    double tnew = tprev + dt;
    tprev = tnew;
    xprev = xnew;
    vprev = vnew;
    // find exact solution to compare approximation with
    double exact_nodamp = harmonic_exact(tnew);
    printf("%g   %g   %g   %g   %g\n", tnew, xnew, vnew, exact_nodamp, acc);
  }
}
```

Run this program with

```
gcc -o nonlinear-pendulum nonlinear-pendulum.c -lm
./nonlinear-pendulum > nonlinear-pendulum.dat
```

Now let's use gnuplot to graph the damped spring:

```
gnuplot
# and at the gnuplot> prompt:
set grid
set title "damped harmonic oscillator"
set xlabel "time"
set ylabel "amplitude"
plot "nonlinear-pendulum.dat" using 1:2 with lines title  "damped oscillator approximate"
replot  "nonlinear-pendulum.dat" using 1:4 with lines title "damped oscillator exact"
```

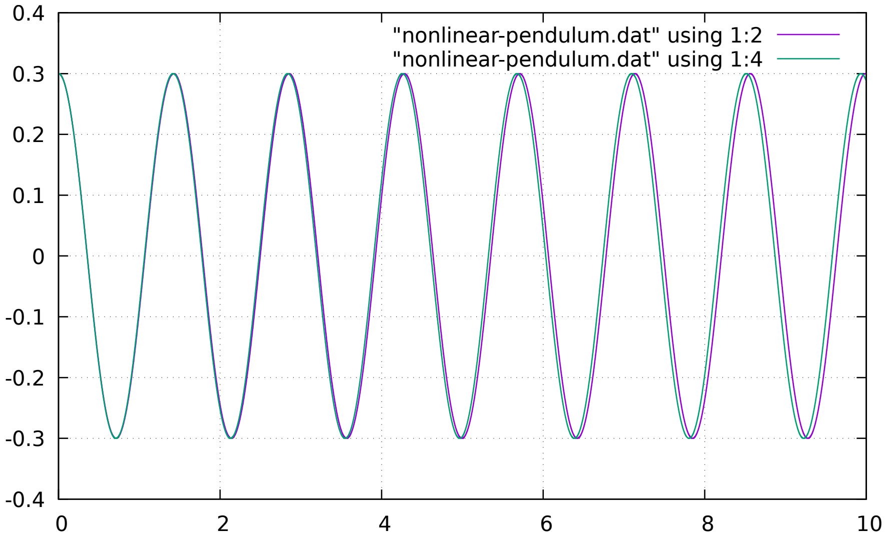
It should look something like this:

Take a careful look at that last plot, and at the source code, specifically the setting of:

```
double theta_0 = 0.3;           /* radians */
```

0.3 radians is a rather small angle, so the approximation $\sin(\theta) \approx \theta$ almost holds at $\sin(0.3) = 0.29552$. But after a few periods you see that the solutions diverge.

If you set `theta_0` to be 0.1 then the approximation will work for a long time. If you set it to be 0.8 you will see the curves diverge quite soon.


"nonlinear-pendulum.dat" using 1:2
"nonlinear-pendulum.dat" using 1:4
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0
2
4
6
8
10

CHAPTER

EIGHTEEN

ECOLOGY

[status: barely-started]

18.1 Motivation, Prerequisites, Plan

As I write this, in April of 2020, it seems like a good opportunity ot get comfortable with some of the equations that come up when we talk about growth. We will look at the growth of a population, or growth of the number of infected humans (which is related to the growth of the population that carries the infection).

Let us start with mosquitoes and West Nile Virus in Texas. Watch this crash course video at:

https://www.youtube.com/watch?v=RBOsqmBQBQk&index=2&list=PL8dPuuaLjXtNdTKZkV_GiIYXpV9w4WxbX

Then have ready this nature paper on basics of ecology:

https://www.nature.com/scitable/knowledge/library/an-introduction-to-population-growth-84225544/

18.2 Factors that come up in modeling population ecology

Table 18.2.1: Factors in population ecology

name	variable	
initial pop:	N	
birth rate:	B	
death rate:	D	
growth rate:	r	= (B - D) / N
predation		
immigration		
emigration		
mates		
food		
space		

18.3 Exponential growth

Spend some time plotting exponentials in gnuplot. Show how they dwarf linear growth, and how you need log scale to compare them.

18.4 History of the human population on earth

Peruse the Wikipedia page on historical population estimates:

https://en.wikipedia.org/wiki/Estimates_of_historical_world_population

and the study at:

https://www.prb.org/howmanypeoplehaveeverlivedonearth/

Spend some time exploring the interactive graphs at:

https://ourworldindata.org/world-population-growth

Expand the title on "All our charts on World Population Growth", and pick the population by country since 1500 and try to understand what areas are exponential.

Then look at the link "World population since 10,000 BCE (OurWorldInData series)".

Download the data for this graph and zoom in on some specific periods. Look at both linear and logarithmic scales.

Following the indications shown in

https://www.nature.com/scitable/knowledge/library/an-introduction-to-population-growth-84225544/

we can look at the table below and seek certain interesting periods in the data.

Table 18.4.1: Periods of interest in human population history

Start	End	What to look for
-10000	-4000	Agricultural revolution
-4000	-600	Early empires
-1000	300	Alexander and Rome
1	300	Imperial Rome
1	1600	Largely steady world population
1200	1400	Medieval black death
1500	present	Modern world
1850	present	Industrial revolution
1900	present	Large scale science

18.5 The logistic function

Although the earth's population as a whole appears to still be in an exponential growth phase, the Pew Research Center predicts that it will flatten by the end of the 21st century:

https://www.pewresearch.org/fact-tank/2019/06/17/worlds-population-is-projected-to-nearly-stop-growing-by-the-end-of-the-century/

This type of function is not exponential growth anymore: it shows exponential growth, but that then slows down and we end up with what is called the Logistic Function:

https://en.wikipedia.org/wiki/Logistic_function

Think of fidget spinners.

18.6 The Lotka-Volterra differential equations

18.7 Further reading

- https://www.youtube.com/watch?v=NYq2078_xqc - Khan Academy video with pleasant intro to cycles and real examples, 5min.
- https://www.youtube.com/watch?v=mFDiiSqGB7M - crash course on predator-prey ecology
- https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations#A_simple_example
- http://www.scholarpedia.org/article/Predator-prey_model
- http://mc-stan.org/users/documentation/case-studies/lotka-volterra-predator-prey.html

Still have to look at articles on 3-way predator-prey:

- http://emis.ams.org/journals/HOA/JAMDS/3/2155.pdf
- chrome-extension://oemmndcbldboiebfnladdacbdfmadadm/https://www.cs.unm.edu/~forrest/classes/cs365/lectures/Lotka-Volterra.pdf
- http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.360.1552&rep=rep1&type=pdf
- http://disi.unal.edu.co/~gjhernandezp/sim/lectures/DeterministicModelsAndChaos/PopulationModels/LotkaVolterra3species.pdf

And how about simple models of a full society collapse?

- https://en.wikipedia.org/wiki/Collapse:_How_Societies_Choose_to_Fail_or_Succeed
- http://necsi.edu/projects/evolution/co-evolution/pred-prey/co-evolution_predator.html
- https://www.ted.com/talks/jared_diamond_on_why_societies_collapse
- https://arxiv.org/abs/1002.0068
- https://faustusnotes.wordpress.com/2014/05/15/mathematical-modeling-of-civilization-collapse/

Cliometrics

- https://en.wikipedia.org/wiki/Cliometrics

CHAPTER

NINETEEN

BIOLOGY – PHYLOGENY

[status: written, but incomplete]

19.1 Motivation, prerequisites, plan

19.1.1 Motivation

One of the most important areas of research in biology is that of phylogenetic analysis. This collection of techniques allows us to build an *evolutionary tree* showing how various species are related.

This type of analysis can also be used in other areas, such as tracing the origin of human spoken languages.

I find phylogenetic analysis to be fascinating because it gives us a sort of "webcam of the gods", a view of the past (which we cannot see) which brought about the present state of things.

19.1.2 Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**

19.1.3 Plan

We will start by looking at a video tutorial of how to build a simple phylogenetic tree by hand. Then we will learn how to use biopython and ete3 packages to construct and visualize trees on that simple problem.

Then we discuss further projects in which we look for data sets to work with, including sets from our own genetic algorithm runs (where we also know the real evolutionary history), human languages (where we do *not* know the real history), and computer programming languages (where we should know most of the real history).

https://cnx.org/contents/24nI-KJ8@24.18:EmlvXoDL@7/Taxonomy-and-phylogeny

19.2 Start with a video and then make a simple table

Start with this Khan Academy tutorial on phylogenetic trees

Then we take their table of traits. Start with the empty table:

Table 19.2.1: An empty trait table which the class could fill together.

Species	Feathers	Fur	Lungs	Gizzard	Jaws
Lamprey					
Antelope					
Sea Bass					
Bald Eagle					
Alligator					

and write it on the board. Then fill out the tree on the board with the class. You can discuss what all these animals are, and look them up if necessary.

The table will end up looking like this:

Table 19.2.2: What the trait table should like like once it is filled.

Species	Feathers	Fur	Lungs	Gizzard	Jaws
Lamprey	no	no	no	no	no
Antelope	no	yes	yes	no	yes
Sea Bass	no	no	no	no	yes
Bald Eagle	yes	no	yes	no	yes
Alligator	no	no	yes	no	yes

Then use the principle of parsimony to create the phylogenetic tree, following the guidelines in the tutorial. The result should look like what you see in Figure **??**.

Discuss the meaning of parsimony as seen in this example. Connect it to Ockham's razor.

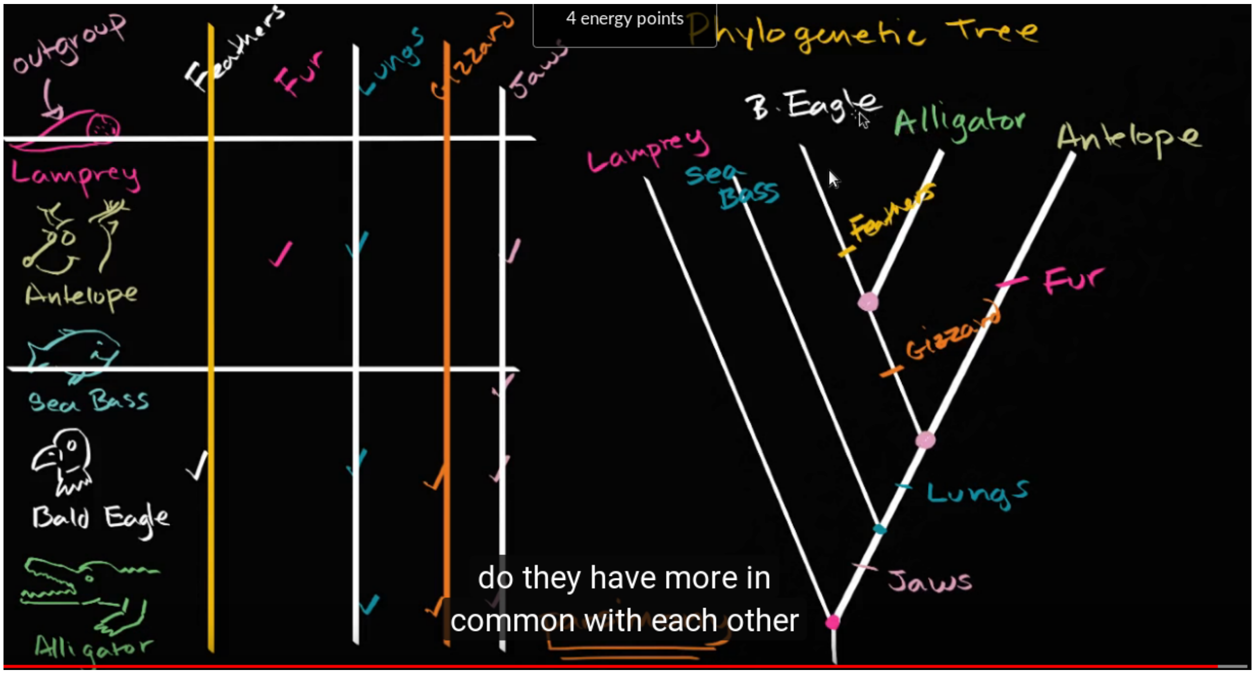
19.3 Terminology

Clades, taxa, species, genotype, phenotype, . . .

The tree of life

19.4 NEW - Installing necessary packages

```
sudo apt install python3-biopython python3-matplotlib
```



do they have more in
common with each other

Figure 19.2.1: The resulting tree from the Khan Academy video example.

19.5 NEW - first steps with biopython

Tutorials are at:

https://taylor-lindsay.github.io/phylogenetics/

and

http://biopython.org/DIST/docs/tutorial/Tutorial.html

Data from opentreeoflife at:

https://tree.opentreeoflife.org/

I tied Streptococcus_mitis_NCTC_12261_ott725 at:

https://tree.opentreeoflife.org/opentree/argus/ottol@175918/Streptococcus

and downloaded the Newick format of the streptococcus subtree at:

https://tree.opentreeoflife.org/opentree/default/download_subtree/ottol-id/175918/Streptococcus

with:

```
wget --output-document subtree-ottol-175918-Streptococcus.tre https://tree.opentreeoflife.org/opentree/default/download_subtree/ottol-id/175918/Streptococcus
```

```
from io import StringIO
from Bio import Phylo
from Bio.Phylo.TreeConstruction import DistanceCalculator
t = Phylo.read(StringIO("((a,b),c);" ), format="newick")
Phylo.draw(t)
```

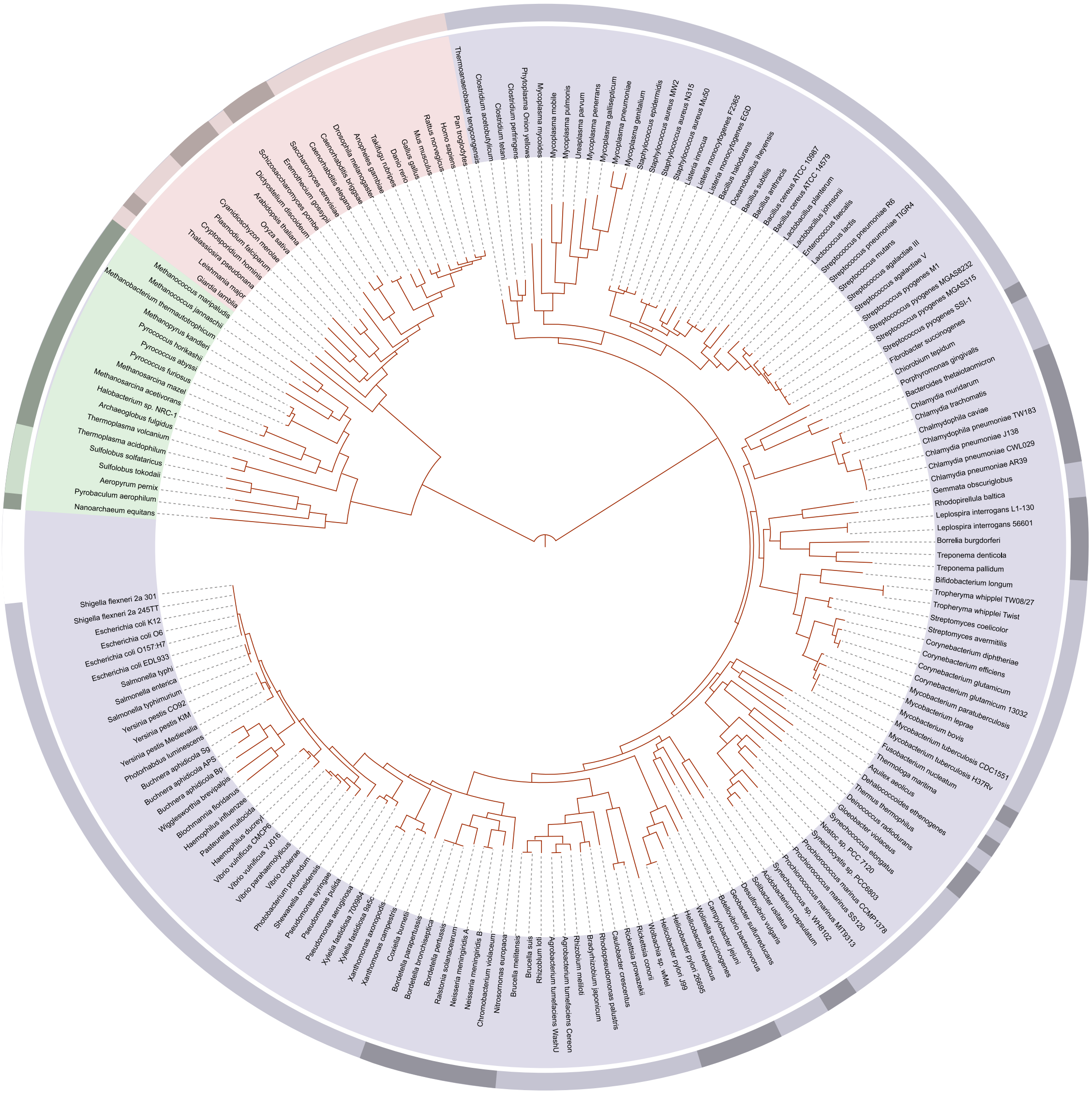


Figure 19.3.1: Hillis's tree of life based on completely sequenced genomes (from the Wikipedia image)

```
import os
import matplotlib
import matplotlib.pyplot as plt
from Bio import Phylo
from Bio.Phylo.TreeConstruction import DistanceCalculator

os.system('wget https://api.opentreeoflife.org/v3/study/ot_2221.tre')
tree = Phylo.read("ot_2221.tre", "newick")
Phylo.draw(tree)
Phylo.draw_ascii(tree)

os.system('wget --output-document subtree-ottol-175918-Streptococcus.tre https://tree.
↪opentreeoflife.org/opentree/default/download_subtree/ottol-id/175918/Streptococcus')
tree = Phylo.read("subtree-ottol-175918-Streptococcus.tre", "newick")
Phylo.draw(tree)
```

19.6 OLD - Installing necessary packages

Follow the instructions at http://etetoolkit.org/download/

```
# Install Minconda  (you can ignore this step if you already have Anaconda/Miniconda)
wget http://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh -O Miniconda3-
↪latest-Linux-x86_64.sh
bash Miniconda3-latest-Linux-x86_64.sh -b -p ~/anaconda_ete/
export PATH=~/anaconda_ete/bin:$PATH;

# Install ETE
conda install -c etetoolkit ete3 ete_toolchain

# Check installation
ete3 build check
```

Now to test it run this simple program. You can even paste it into the python3 interpreter.

```
from ete3 import Tree
t = Tree( "((a,b),c);" )
t.render("mytree.png", w=183, units="mm")
```

19.7 Preparing a tree by hand

Now let us prepare a tree where we input it ourselves. The format is like what we saw in the example above: the tree is made with the call `Tree( "((a,b),c);" )`

But we will make a slightly more interesting tree, the one we worked out in Section **??**. To do so enter the program in Listing **??**.

Listing 19.7.1: Program which makes a phylogenetic tree from a simple example tree.

```
#! /usr/bin/env python3
from ete3 import Tree

out_fbase = 'tree-sample-animals'

# t = Tree( "((a,b),c);" )
t = Tree( "((((Eagle,Alligator),Antelope),Sea Bass),Lamprey);" )
for format in ('png', 'svg'):
    out_fname = out_fbase + '.' + format
    print('writing output to', out_fname)
    t.render(out_fname, w=183, units="mm")
```

You can adjust the look of this tree. See the discussion in:

http://etetoolkit.org/docs/3.0/tutorial/tutorial_drawing.html

we can go through that and adjust our styles a bit and see how our tree looks.

19.8 Inferring a tree

The problem with the program in Listing **??** is that it prepares the tree, which you can view with your favorite PNG or SVG file viewer. But it does not *find* the tree. That is our next goal.

So we want to find the most likely evolutionary tree that would yield the result we see in the Table **??**. This process is called *inferring* the phylogenetic tree from the table of characteristics.

19.8.1 An example input file provided by ete3

Following the ete cookbook at http://etetoolkit.org/cookbook/ete_build_basics.ipynb

let us try:

```
# find some place to download NUP62.aa.fa
$ mkdir phylo
$ cd phylo
$ locate NUP62.aa.fa
/home/markgalassi/anaconda_ete/lib/python3.6/site-packages/ete3/test/test_ete_build/
↪NUP62.aa.fa
/home/markgalassi/anaconda_ete/pkgs/ete3-3.1.1-pyhf5214e1_0/site-packages/ete3/test/test_
↪ete_build/NUP62.aa.fa
$ cp /home/markgalassi/anaconda_ete/lib/python3.6/site-packages/ete3/test/test_ete_build/
↪NUP62.aa.fa ./
$ cat NUP62.aa.fa | head -n15
$ ete3 build -w standard_fasttree -a NUP62.aa.fa -o NUP62_tree/ --clearall
$ ls NUP62_tree/ -ltr
$ geeqie NUP62_tree/clustalo_default-none-none-fasttree_full &
```

Adapting our table to the `.fa` file format, we need a name for each organism, and an encoding for the traits. The ete3 team's example has this line, for example:

```
>Phy004Z0OU_MELUD
MSQFSFGTGGGFTLGTSGTAASTAATGFSFSSPAGSGGFGLGSAAPAAGSSSQSSGLFSF
SRPAATAAQPGGFSFGTAGTSSAAPAASVFQLGANAPKLSFGSSSATPATGITGSFTFGS
SAPTSAPSSQAAAPGFVFGSAGTSSTAQAGTTAGFTFSSGTTTQAGAGSLSMGAAVPQTA
PTGLSFGAAPAAAATSAATLGAATQPAAPFSLGGQSTATATVSTSTSSGPALSFGAKLGV
TSTSAATASTSTTSVLGSTGPTLFASVASSAAPASSTTTGLSLGAPSTGTASLGTLGFGL
KAPGTTSAATTSTATGTTTASGFALNLKPLTTTGATGAVTSTAAITTTTSTSAPPVMTYA
QLESLINKWSLELEDQEKHFLHQATQVNAWDQTLIENGEKITSLHREVEKVKLDQKRLDQ
ELDFILSQQKELEDLLTPLEESVKEQSGTIYLQHADEEREKTYKLAENIDAQLKRMAQDL
KDITEHLNTSRGPADTSDPLQQICKILNAHMDSLQWIDQNSAVLQRKVEEVTKVCESRRK
EQERSFRITFD
```

and we could have something like:

Listing 19.8.1: Table of traits for the animals we discussed earlier FIXME put table cross-reference.

```
>Lamprey
NNNNN
>Antelope
NYYNY
>Sea_Bass
NNNNY
>Bald_Eagle
YNYNY
>Alligator
NNYNY
```

Put this information into a file called `simple-animals.fa` and use the `ete3 build` command to infer a phylogenetic tree for it:

```
ete3 build -w standard_fasttree -a simple-animals.fa -o simple-animals_tree/ --clearall
```

This program will put graphical output files in the directory `simple-animals_tree/clustalo_default-none-none-fasttree_full/` and you can view the .png, .svg and .pdf files there.

The output shows the same family links that we obtained by hand, but the tree looks different because the *root* is placed differently. FIXME: must find the correct invocation of ete3 to root the tree parsimoniously.

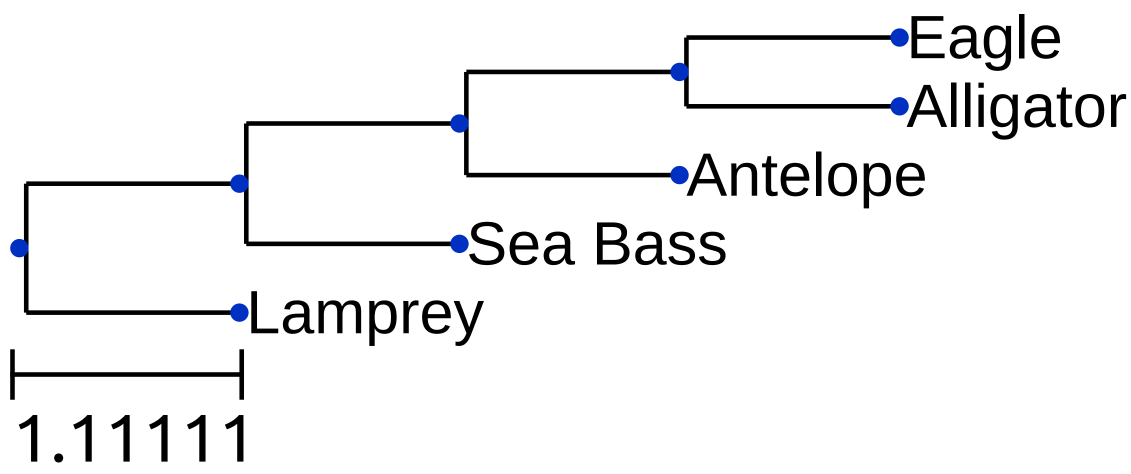
You can also view this tree as ascii with:

```
ete3 view -t simple-animals_tree/clustalo_default-none-none-fasttree_full/simple-animals.
↪fa.final_tree.nw
```

We can process it further with a few python instructions:

```
from ete3 import PhyloTree
tree = PhyloTree("simple-animals_tree/clustalo_default-none-none-fasttree_full/simple-
↪animals.fa.final_tree.nw")
tree.link_to_alignment("simple-animals_tree/clustalo_default-none-none-fasttree_full/
↪simple-animals.fa.final_tree.used_alg.fa")
tree.set_outgroup('Lamprey')
tree.render("%%inline")
tree.render("simple-animals-rooted-outgroup.svg", w=183, units="mm")
```

(continues on next page)



(continued from previous page)

```
tree.render("simple-animals-rooted-outgroup.png", w=183, units="mm")
print(tree)
```

This will save svg and png formatted views of the tree, as well as showing you an ascii representation.

The documentation on ete3 show a dizzying variety of tree styles. The key is to define a tree style using the Python TreeStyle class, and then use it as a parameter to how we visualize our tree.

Here are a couple of of examples, continuing from the previous code. The first, taken from the ETE tutorial, shows a circular tree in 180 degrees.

```
# [this uses the tree built in the previous code block]
from ete3 import Tree, TreeStyle
ts = TreeStyle()
ts.show_leaf_name = True
ts.mode = "c"
ts.arc_start = -180 # 0 degrees = 3 o'clock
ts.arc_span = 180
tree.render("simple-animals-circular.svg", tree_style=ts)
tree.render("simple-animals-circular.png", tree_style=ts)
tree.show(tree_style=ts)
## note that instead of tree.show(), which opens a live tree
## browser, you could use tree.render() to save it to a file
```

Another example shows our tree as a bubble tree map:

19.9 Other sequence analysis resources

Berkeley evolution course. 7 organisms and 7 features: https://evolution.berkeley.edu/evolibrary/article/phylogenetics_07

Cute with ladybugs, but just 6 elements and 7 features: https://bioenv.gu.se/digitalAssets/1580/1580956_fyltreeeng.pdf

Another video giving step-by-step for building a tree by hand: https://www.youtube.com/watch?v=09eD4A_HxVQ

19.10 Linguistics datasets

First discuss the issues of generating a string representation of language features. One example of the issues involved is given in this article:

https://brill.com/view/journals/ldc/3/2/article-p245_4.xml?lang=en

where they discuss how to align the English "horn" with the latin "kornu". This then allows you to define a "genetic distance" between the same word in two different languages. One such measure is the "Levenshtein normalized distance" (LDN), which takes values between 0 and 1.

This can then be used with the ASJP (Automated Similarity Judgement Program) https://en.wikipedia.org/wiki/Automated_Similarity_Judgment_Program database which is based on a word list. The database is at https://asjp.clld.org/

Download the database from

https://asjp.clld.org/download

and look at the listss18.txt file and see how languages we know (English, Italian, Spanish, Russian) are represented.

Look at the results in WorldLanguageTree001.pdf

One oft-used list is the the Swadesh list https://en.wikipedia.org/wiki/Swadesh_list which has 100 terms. Some of these are "I", "you", "we", "this", "that", "person", "fish", "dog", "foot", "hand", "sun", "mountain", various basic colors, and so forth. There is also an abbreviated 35-word list. The ASJP uses a 40-word list, similar to the Swadesh list.

Each part of a word gets an ASJP code and an IPA (International Phonetic Alphabet) designation of how it's pronounced.

19.10.1 lingpy.org

Go through the tutorial, starting at:

http://www.lingpy.org/tutorial/index.html

install with

```
pip3 install lingpy
```

then simple examples at:

http://www.lingpy.org/examples.html

then the workflow tutorial at:

http://www.lingpy.org/tutorial/workflow.html

and the cookbook at:

https://github.com/lingpy/cookbook

19.10.2 elinguistics.com

Overall language evolutionary tree: http://www.elinguistics.net/Language_Evolutionary_Tree.html You can follow the links to some detailed discussion of timelines at http://www.elinguistics.net/Language_Timelines.html as well as in-depth discussion of encoding and language comparison. In particular you will find various sounds for key encoding words at http://www.elinguistics.net/Compare_Languages.aspx?Language1=English&Language2=German&Order=Details

The choice of "basis words" to use as "genetic markers" is described at http://www.elinguistics.net/Lexical_comparison.html and the continuing pages http://www.elinguistics.net/Sound_Correspondence.html and back to the example of English to German at http://www.elinguistics.net/Compare_Languages.aspx?Language1=English&Language2=German&Order=Details

19.10.3 Others

https://en.wikipedia.org/wiki/Tree_model#Perfect_phylogenies

https://en.wikipedia.org/wiki/Tree_model

https://en.wikipedia.org/wiki/Language_family

https://en.wikipedia.org/wiki/Tree_model#CITEREFNakhleh2005

https://www.theguardian.com/education/gallery/2015/jan/23/a-language-family-tree-in-pictures

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3049109/

https://linguistics.stackexchange.com/questions/14905/is-there-a-phylogenetic-tree-for-all-known-languages

https://glottolog.org/

https://glottolog.org/resource/languoid/id/stan1293

https://glottolog.org/glottolog/family

https://www.ethnologue.com/browse/families

https://glottolog.org/resource/languoid/id/macr1271

https://science.sciencemag.org/content/323/5913/479

Look at the PDF for this paper on phylogeny of polynesian languages:

https://www.researchgate.net/publication/23933879_Language_Phylogenies_Reveal_Expansion_Pulses_and_Pauses_in_Pacific_Settlement

And this one with very nice-looking pictures of Japonic language evolution and some discussion of word differentiation.

https://royalsocietypublishing.org/doi/full/10.1098/rspb.2011.0518

Natural language processing with Python and NLTK:

http://www.nltk.org/book/

19.11 Evolution of programming languages

https://www.i-programmer.info/news/98-languages/8809-the-evolution-of-programming-languages.html

https://royalsocietypublishing.org/doi/full/10.1098/rsif.2015.0249

CHAPTER

TWENTY

RECURSION

[status: partially-written]

20.1 Motivation, prerequisites, plan

Recursion comes up in many very important computer algorithms, to the point where it's reasonable to say that it is an essential tool for most programmers. When an algorithm can be expressed recursively it almost feels like we're cheating in how we implement it: we get this complex result from two very simple statements!

The only prerequisite for this chapter is the 10-hour *Serious Computer Programming for Youth* course.

Our plan is to first try to get a feeling for recursion with some visual examples. Then we state the two key parts of a recursive definition (and hence of a recursive algorithm): the *recurrence relation* and the *initial condition*. With this under our belts we take on examples from math and other areas.

20.2 Visual examples

- Point a webcam at the screen.
- https://i.stack.imgur.com/0DaD5.jpg (or other paintings – Escher?).
- Break sections from broccoli.
- Draw koch snowflake or Sierpinski gasket.

20.3 Word examples

History of the Liar's Paradox

From Epimenides to Quine

Epimenides said "all Cretans are Liars". The modern version is "This sentence is a lie."

- "This sentence is false."
- "The next sentence is true. The previous sentence is false."
- Read this sentence and do what it says twice.

- Dialogs from Godel, Escher, Bach
- Quining: "yields falsehood when preceded by its quotation" yields falsehood when preceded by its quotation. https://en.wikipedia.org/wiki/Quine%27s_paradox
- Russell's Paradox. https://en.wikipedia.org/wiki/Russell%27s_paradox
- Aristotle's "law of excluded middle". https://en.wikipedia.org/wiki/Law_of_excluded_middle

20.4 Components of a recursive definition

A recursive series is defined by two components:

Recurrence relation
Specifies what the *next* element of a sequence is, based on the previous one. If we are talking about a numbered sequence, then this means defining the n^{th}element in terms of the $(n-1)^{\text{th}}$element (and maybe $n-2$ and earlier elements as well).

Initial condition
You can't go on applying the recurrence relation forever, so at some point you have to stop. We do this by specifying an initial condition. For example we can set an element to have a clear value when $n = 0$ or $n = 1$.

Let us look at some examples of recursive definitions in simple math.

20.4.1 Simple math

Define power with recursion:

$$3^5 = 3 * 3^4 = 3 * (3 * 3^3) = 3 * (3 * (3 * 3^2)) = 3 * (3 * (3 * (3 * 3)))$$

which suggests this generalization:

$$x^n = 1 \qquad \text{when } n = 0 \tag{20.4.1}$$
$$x^n = x \qquad \text{when } n = 1 \tag{20.4.2}$$
$$x^n = x \times x \qquad \text{when } n = 2 \tag{20.4.3}$$
$$x^n = x \times x^{n-1} \qquad \text{when } n > 2 \tag{20.4.4}$$

In this you can see that x^n is defined recursively with recurrence relation $x^n = x \times x^{n-1}$ and initial condition $x^0 = 1$.

Factorials are defined like this:

$$2! = 2 \times 1 = 2 \tag{20.4.5}$$
$$3! = 3 \times 2 \times 1 = 6 \tag{20.4.6}$$
$$\dots \tag{20.4.7}$$
$$n! = n \times (n-1) \times (n-2) \cdots \times 2 \times 1 \tag{20.4.8}$$

which lends itself to a very simple recursive definition:

$$n! = 1 \qquad \text{when } n = 1 \tag{20.4.9}$$
$$n! = n \times (n-1)! \qquad \text{when } n > 1 \tag{20.4.10}$$

This is very easily implemented in Python:

```
def factorial_recursive(n):
    if n == 1:
        return 1
    else:
        return n * factorial_recursive(n-1)
```

20.4.2 Programming simple math recursion

In Listing **??**: you can see a simple program which calculates x^n. Go ahead and try it out to calculate powers.

Listing 20.4.1: Program which calculates powers using recursion.

```
#! /usr/bin/env python3

"""
A demonstration of how to calculate x^n using recursion.
"""

def main():
    x = float(input('give x: '))
    n = float(input('give n: '))
    result = power(x, n)
    print('%g^%d is: %g' % (x, n, result))

def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)

main()
```

Exercises

Exercise 20.1 Write an analogous program that calculates n! You may use Listing **??** as a guide.

Exercise 20.2 The *fibonacci numbers*, which appear in many beautiful manifestations of nature, are defined recursively:

$$fib(n) = 1 \qquad \text{when } n = 0 \textit{ or } n = 1 \tag{20.4.11}$$

$$fib(n) = fib(n-1) + fib(n-2) \qquad \text{when } n > 1 \tag{20.4.12}$$

Now write a program that calculates $fib(n)$. You may once again use Listing **??** as a guide.

20.4.3 Recursion with data structures

The list is a very important data structure, and it often lends itself to surprisingly simple recursive algorithms.

A simple example is to *reverse* a list in python. A list with just one element `l = ['dude']` will have an obvious reverse, which is just itself. And if you have reversed a list with n elements, then to do a list with one more element is easy. Start out with:

```
l = [2.5, 17, 'dude']
```

and imagine that you have reversed all but the last element:

```
# this is not ready to run!
l = [2.5, 17, 'dude']
# we have somehow reversed the first two elements
l_reversed_without_last = [17, 2.5]
# now we complete the task by putting the last element at the start
l_reversed = [l[-1]] + l_reversed_without_last
```

From this pseudo-code *(that will absolutely not work!)* you can get an idea for a recursive algorithm:

Recurrence relation
reverse(list) = [last_element] + reverse(list_without_last_element)

Initial condition
reverse(list_with_just_one_element) = that_list_with_just_one_element

Remembering that in Python you can get the last element of a list `l` with `l[-1]`, and the rest of the list with `l[:-1]`, we can write an inline function to do this:

```
def reverse_list_recursive(l):
    if len(l) == 1:
        return l
    else:
        return reverse_list_recursive(l[1:]) + [l[0]]
```

```
# now run it with:
reverse_list_recursive([2.5, 17, 'dude'])
```

```
# do some numbers with:
reverse_list_recursive(list(range(20)))
```

```
# now do a much longer list
reverse_list_recursive(list(range(200)))
```

20.5 Visualizing what the recursion is doing

We clearly agree that the magic of recursion works, but it can be frustrating to not have an intuition of the intermediate steps. Let us modify the function so that it prints how it's putting together the pieces of the lists in the intermediate stages.

Listing 20.5.1: Reverse a list with a recursive algorithm, breaking it into `l[0]` and `l[1:]`. This prints some information on the calls to the recursive algorithm.

```
def reverse_list_recursive_pivot_0(l):
    """Reverse the list l using a recursive algorithm based on breaking it
    down into l[0] and l[1:]
    """
    if len(l) == 1:
        return l
    else:
        # print information on how the recursion relation is going
        print(f'  RECURSION: reverse({l[1:]}) + {[l[0]]}')
        return reverse_list_recursive_pivot_0(l[1:]) + [l[0]]


# now try it out with a few examples of lists of numbers
for listlen in range(6):
    print('------- reversing list', list(range(listlen+1)), '------')
    result = reverse_list_recursive_pivot_0(list(range(listlen+1)))
    print('RESULT:', result)
    print()
```

20.6 Towers of Hanoi

The Towers of Hanoi is a puzzle in which you have three pegs and a pile of discs on one of them. The discs always *must* be piled with bigger discs below smaller discs. The goal is to move the discs from their initial peg to another peg, using the extra peg if you need to.

Listing 20.6.1: Recursive solution to the Towers of Hanoi game

```
#! /usr/bin/env python3

"""
A demonstration of how to solve the Towers of Hanoi game using
recursion
"""


def main():
    n_disks = int(input('how many discs? '))
    move_tower(n_disks, 'A', 'B', 'C')

def move_tower(height, from_pole, to_pole, interim_pole):
    if height > 0:
```

(continues on next page)

(continued from previous page)

```
        move_tower(height-1, from_pole, interim_pole, to_pole)
        move_disk(from_pole, to_pole)
        move_tower(height-1, interim_pole, to_pole, from_pole)

def move_disk(from_pole, to_pole):
    print('moving disk from', from_pole, 'to', to_pole)

main()
```

Now look at the program in Listing **??**. This is a surprisingly short program because the information about the state of the pegs and discs is not in any of the program variables! It's all stored in function stack as part of recursive calls.

Exercise 20.1 Count how many disc movements are made in total to solve the puzzle, and plot that as a function of how many discs you picked for that run.

Exercise 20.2 Find a way to intercept the program to draw (in simple ascii art, or with a canvas as shown in Section **??**) the intermediate stages of the solution.

20.7 Should we really use recursion in programming?

We saw in Section **??** and we will see again in Section **??** that some problems are expressed very simply using recursive algorithms. Should we always look for recursive solutions?

The trade-offs come when you are dealing with very high numbers of recursive calls. If you imagine yourself reciting this in your head:

> Seven factorial is seven times six factorial which is seven times six times five factorial which is … which is seven times six times five times four times three times two times one times zero factorial; that last one is one so I can now finally multiply them all together: seven times six times five times four times three times two times one times one which is fivethousand and fourty.

you can see that you have to keep a lot of stuff in mind. The nonrecursive algorithm:

```
def factorial_nonrecursive(n):
    assert(n > 0)
    result = 1
    for i in range(1, n+1):
        result = result * i
    return result
```

never juggles more than two numbers at once in its memory.

Another problem with recursive list-processing algorithms is that they often have to shuffle around copies of the list data. The list reversing algorithm in Listing **??** shows how list slices like `l[1:]` cause most of the list data to be copied in memory.

CHAPTER

TWENTYONE

PROGRAMMING TOPICS: SORTING

[status: content-mostly-written]

21.1 Motivation, prerequisites, plan

There are several reasons to spend some time on sorting. Apart from the usefulness of sorting a list of data, we will study it because:

- It illustrates programming techniques.
- We will go from an intuitive understanding of sorting to how that can be expressed as an *algorithm*.
- It will give us a simple and yet rich example of how to study the *computational complexity* of an algorithm: how the execution time grows with the size of the data we work with.

Prerequisites:

- The 10-hour “serious programming” course.

Our plan is to start with an intuitive discussion of what sorting is, and to distill an algorithm out of that. We will look in to *sorted insertion*, and then into the *bubblesort* algorithm.

21.2 Experiment: a game of cards

Start by dealing out a game of cards. We can start with Poker, or bridge if you and the other students have already seen it.

When you receive your hand you should:

1. Describe their algorithm to your neighbor in class.
2. Compare what happens when you sort 4 cards to when you sort 8 or 13 cards.
3. Take notes on paper on how you sorted your hand so as to see all the features in it.

21.3 Intuition to algorithm on card sorting

Take your description of the algorithm and try to write a python function to implement that sorting. We will do so in the framework of a small program that is ready to go as soon as you drop in your sorting function.

21.4 Writing up the algorithm in Python

I show here a simple algorithm for sorting, which might look a lot like what you have come up with. The Python function is called `sort_myveryown()`

```python
def sort_myveryown(l):
    """This function implements my very own sorting algorithm, which
    is basically an insertion sort.  It takes the list l, sorts
    it, and returns the sorted result."""
    ## iterate through the whole list, skipping element 0
    for i in range(1, len(l)):
        ## having fixated on element i, look at all the ones
        ## *before* i and see if they are bigger and should
        ## be placed *after* i
        for j in range(i):
            if l[i] < l[j]:
                ## this is the case where l[j] is bigger.  since
                ## all the j come *before* the i this means that
                ## the sort is incorrect and we must swap i and j
                # print('SWAP: %d/%d  <--> %d/%d' % (i, l[i], j, l[j]))
                l[j], l[i] = l[i], l[j]
    return l
```

Look at the program in Listing **??** and put your own code in for the function `sort_myveryown()`. You can look up "insertion sort" in wikipedia for a very simple description of the algorighm, and their "pseudocode" (a description of the algorithm that is not in any specific programming language) can be translated to Python quite easily. If you feel that your procedure was different from insertion sort then adapt the code to match what you did.

Listing 21.4.1: sort_frame.py – Framework for trying out sorting algorithms. You can drop your own into the function sort_myveryown()

```python
#! /usr/bin/env python3

"""This is a framework into which you can drop your Python sorting
routine.  The function sort_myveryown() is empty (it just returns the
original list) and you can use it to put in your own sorting
function.

Two other functions are provided for comparison: one is a
hastily-written bubblesort routine, the other is the built-in python
list sorting method.

"""

import random
import inspect
```

(continues on next page)

(continued from previous page)

```

def main():
    N = 20
    run_tests(N)

def run_tests(N):
    l_presorted = list(range(N))

    ## l_random will have random numbers with a non-fixed seed
    random.seed(None)
    l_random = [0]*N
    for i in range(len(l_random)):
        l_random[i] = random.randint(0, 99)
    random.seed(1234)

    ## l_random_fixed will always have the same random numbers
    l_random_fixed = [0]*N
    for i in range(len(l_random_fixed)):
        l_random_fixed[i] = random.randint(0, 99)

    ## l_turtle is designed to do quite poorly with some algorithms:
    ## it has a small value at the end
    l_turtle = list(range(1, N-1))
    l_turtle.append(0)

    list_name_dict = {'l_presorted' : l_presorted,
                      'l_random_fixed' : l_random_fixed,
                      'l_random' : l_random,
                      'l_turtle' : l_turtle}

    for algo in (sort_quicksort, sort_python_builtin, sort_bubble, sort_myveryown):
        print('algorithm: %s' % algo.__name__)
        for which_list in list_name_dict.keys():
            print('    list: %s' % which_list)
            l_before = list_name_dict[which_list]
            l_sorted = algo(list_name_dict[which_list])
            print('              ', l_before, ' ----> ', l_sorted)

def sort_myveryown(l):
    ## FIXME: must insert my very own sorting function here
    return l

def sort_bubble(l):
    l2 = l[:]
    for i in range(len(l)):
        for j in range(len(l)-1):
            if l2[j] > l2[j+1]:
                l2[j], l2[j+1] = l2[j+1], l2[j]
    return l2


def sort_quicksort(l):
```

(continues on next page)

(continued from previous page)

```
    l2 = l[:]
    do_quicksort(l2, 0, len(l)-1)
    return l2

def do_quicksort(l, low, high):
    if low < high:
        p = do_qsort_partition(l, low, high)
        do_quicksort(l, low, p-1)
        do_quicksort(l, p+1, high)

def do_qsort_partition(l, low, high):
    pivot = l[high]
    i = low-1
    for j in range(low, high):
        if l[j] < pivot:
            i = i + 1
            l[i], l[j] = l[j], l[i]
    if l[high] < l[i+1]:
        l[i+1], l[high] = l[high], l[i+1]
    return i+1

def sort_python_builtin(l):
    l2 = l[:]
    l2.sort()
    return l2

main()
```

21.5 Profiling the algorithm

21.5.1 Modify the program to print information

The term “profiling” is used in software to mean “figuring out how much time a computer program spends in each of its functions. This is an important tool to figure out how to improve performance.

As you have seen, the two operations which are called repeatedly in a sorting algorithm are *comparison* and *swap*. This means that the most important part of our profiling work will be to count how many times we compare two values in the list and how many times we swap two values in the list. Each algorithm will do differently on those two issues.

The program in Listing **??** gives a “framework” in which you can drop your own function and it will count how many times it does a comparison or a swap. It then prints out the results in a form that can be easily plotted.

The way the program does this is to have two functions: `increment_comparisons()` and `increment_swaps()`. We will call these functions every time we do a comparison or a swap.

The program `sort_frame_profiling.py` then prints out the number of comparisons and number of swaps, as well as the size of the list, for many different list sizes and for all the algorithms we use.

Listing 21.5.1: sort_frame_profiling.py – Framework for profiling sorting algorithms. You can drop your own into the function sort_myveryown() and add the increment_swaps() and increment_comparisons() calls to profile the complexity of each algorithm. Download it by clicking here: `sort_frame_profiling.py`

```python
#! /usr/bin/env python3

"""This is a framework for putting in your own sorting function, but
it also has hooks for profiling the sorting function by counting the
number of comparisons and swaps.

"""

import random
import sys

# count_dict = {}

comparisons = 0
swaps = 0

def main():
    ## see if the user gave a command line argument for the list length
    if len(sys.argv) > 1:
        N_MAX = int(sys.argv[1])
    else:
        N_MAX = 100
    for N in range(N_MAX):
        run_sort_algorithms(N)

def run_sort_algorithms(N):
    l_presorted = list(range(N))

    ## l_random will have random numbers with a non-fixed seed
    random.seed(None)
    l_random = [0]*N
    for i in range(len(l_random)):
        l_random[i] = random.randint(0, 100)

    ## l_random_fixed will always have the same random numbers
    random.seed(1234)
    l_random_fixed = [0]*N
    for i in range(len(l_random_fixed)):
        l_random_fixed[i] = random.randint(0, 100)
    random.seed(None)           # return the seed to be None

    ## l_turtle is designed to do quite poorly with some algorithms:
    ## it has a small value at the end
    l_turtle = list(range(1, N-1))
    l_turtle.append(0)

    ## here is the list of names of initial list layout, mapped to the
```

(continues on next page)

(continued from previous page)

```
    ## actual lists
    list_name_dict = {'l_presorted' : l_presorted,
                      'l_random_fixed' : l_random_fixed,
                      'l_random' : l_random,
                      'l_turtle' : l_turtle}

    for algo in (sort_python_builtin, sort_bubble, sort_myveryown, sort_quicksort):
        run_sort_single_algorithm(algo, N, list_name_dict)

def run_sort_single_algorithm(algo, N, list_name_dict):
    reset_stats(list_name_dict)
    # print('algorithm: %s' % algo.__name__)
    for which_list in list_name_dict.keys():
        # print('list: %s' % which_list)
        l_before = list_name_dict[which_list]
        l_sorted = algo(list_name_dict[which_list], which_list)
        # print('              ', l_before, ' ----> ', l_sorted)
        # print_stats(N, algo.__name__, which_list)
    print_stats(N, algo.__name__, list_name_dict)

def sort_myveryown(l, list_type):
    ## FIXME: must insert my very own sorting function here
    return l

def sort_bubble(l, list_type):
    l2 = l[:]
    for i in range(len(l)):
        for j in range(len(l)-1):
            increment_comparisons(list_type)
            if l2[j] > l2[j+1]:
                increment_swaps(list_type)
                l2[j], l2[j+1] = l2[j+1], l2[j]
    return l2

def sort_quicksort(l, list_type):
    l2 = l[:]
    do_quicksort(l2, 0, len(l)-1, list_type)
    return l2

def do_quicksort(l, low, high, list_type):
    if low < high:
        p = do_qsort_partition(l, low, high, list_type)
        do_quicksort(l, low, p-1, list_type)
        do_quicksort(l, p+1, high, list_type)

def do_qsort_partition(l, low, high, list_type):
    pivot = l[high]
    i = low-1
    for j in range(low, high):
        increment_comparisons(list_type)
        if l[j] < pivot:
            i = i + 1
```

(continues on next page)

(continued from previous page)

```
            increment_swaps(list_type)
            l[i], l[j] = l[j], l[i]
    increment_comparisons(list_type)
    if l[high] < l[i+1]:
        increment_swaps(list_type)
        l[i+1], l[high] = l[high], l[i+1]
    return i+1

def sort_python_builtin(l, list_type):
    """Use the built-in sorting function provided by Python.  Note that
    since we don't write the innards of this function, we cannot keep
    track of how many comparisons and swaps it does.  We do know that
    Python's built-in list.sort() and sorted() functions use the
    Timsort algorithm which is a modified version of merge sort which
    uses insertion sort to arrange the list of items into conveniently
    mergeable sections.  An exercise in the text discusses figuring out
    how to count comparisons in this algorithm."""
    return sorted(l)

def reset_stats(list_name_dict):
    """Resets the counts of comparisons and swaps."""
    global comparisons
    global swaps
    comparisons = {l_type: 0 for l_type in list_name_dict.keys()}
    swaps = {l_type: 0 for l_type in list_name_dict.keys()}

def increment_comparisons(list_type):
    """Increment the counter for the number of comparisons of this
    type of list."""
    global comparisons
    comparisons[list_type] += 1

def increment_swaps(list_type):
    """Increment the counter for the number of swaps of this type
    of list."""
    global swaps
    swaps[list_type] += 1

def print_stats(N, algo_name, list_name_dict):
    """Print a line with statistics on how this algorithm performs for the
    various lists with length N"""
    global comparisons
    global swaps
    list_types = sorted(list_name_dict.keys())
    ## open the file to write out this data
    fname = algo_name + '.out'
    if N == 0:
        ## first time around we zero out the file and write a header line
        f = open(fname, 'w')
        print('Starting to write to file %s' % fname)
        f.write('## ALGO: %s\n' % algo_name)
        f.write('## COMMENT: columns are "iter", "number-of-comparisons",\n')
```

(continues on next page)

(continued from previous page)

```
        f.write('## COMMENT: "number-of-swaps" for various types of lists\n')
        f.write('## iter')
        for l_type in list_types:
            f.write('   %s  ' % l_type)
        f.write('\n')
    else:
        f = open(fname, 'a')
    f.write('%5d' % N)
    for l_type in list_types:
        # print('ALGO_%s--LIST_%s--comparisons--swaps: %d %d %d'
        #       % (algo_name, l_type, N, comparisons[l_type], swaps[l_type]))
        f.write('   %5d  %5d' % (comparisons[l_type], swaps[l_type]))
    f.write('\n')               # finish this line of data

    f.close()

main()
```

21.5.2 Run the program and make plots

When you run

```
mkdir sort-stats            ## let's be tidy abogut where are files are
cd sort-stats
python3 sort_frame_profiling.py
```

you will notice that it takes a while to run (depending on the value of `N_MAX` in `main()`), and when it is done you will have a few files called `sort_ALGO.out` for each of the algorithms.

You don't have to wait for it to finish! You can open another terminal and type:

```
cd sort-stats
ls -lsat | head             ## look at recently modified files
wc sort_*.out               ## see how many lines each file has
head sort_*.out             ## look at the start of each file
tail sort_*.out             ## look at the end of each file
```

You can repeat that `tail` command several times as the run goes on and you will get a feeling for the progress being made.

You can also make plots of the performance of the algorithms. Run gnuplot:

```
$ gnuplot
```

and in gnuplot run the following instructions:

Listing 21.5.2: Plot bubblesort and quicksort timing.

```
set grid
set xlabel 'N (size of list)'
set ylabel 'number of operations'
plot 'sort_bubble.out' using 1:4 with lines title 'bubble/random/comparisons', \
 'sort_bubble.out' using 1:5 with lines title 'bubble/random/swaps', \
```

(continues on next page)

(continued from previous page)

```
'sort_quicksort.out' using 1:4 with lines title 'quicksort/random/comparisons', \
'sort_quicksort.out' using 1:5 with lines title 'quicksort/random/swaps'
```

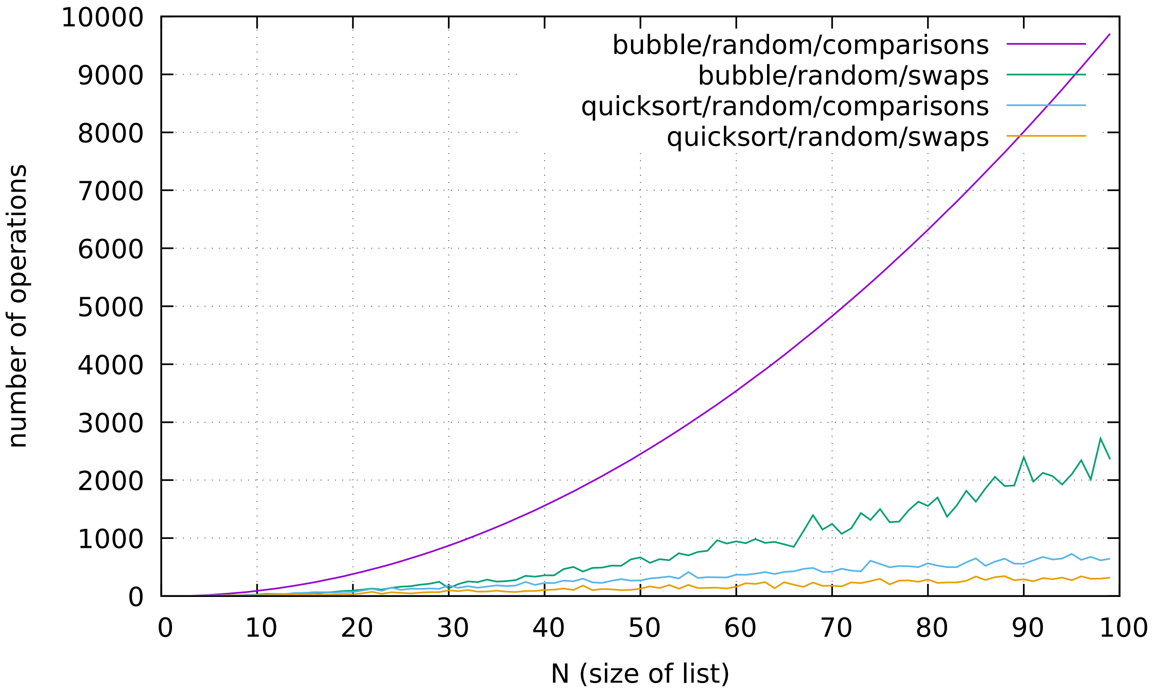


Figure 21.5.1: The performance of bubblesort and quicksort on a random initial list. Note that bubblesort operations grow as n^2 while quicksort operations grow as $n \times log(n)$, which is a much slower growth. This kind of plot shows how the number of operations in an algorithm depends on the size of the problem. This dependence is called the "computational complexity" of an algorithm.

21.5.3 How do we understand these plots?

[TODO] Make some plots in gnuplot of `x**2` and `x log(x)`, then introduce two bits of terminology: "computational complexity" and "asymptotic behavior".

21.5.4 Exercises

Exercise 21.1 Modify `sort_frame_profiling.py` to count the number of *comparisons* made by Python's built-in sort function. One way to do this is to use an argument to `l.sort(my_compare)` which replaces Python's built-in comparison function. You could then write your own `my_compare()` function which would call `increment_comparison(l_type)`.

Exercise 21.2 Research whether it is possible to count the number of *swaps* made by Python's built-in sort function.

21.6 Computational complexity

Make plots versus `N` and compare to N^2 and $Nlog(N)$

21.7 Further reading

The wikipedia articles on various sorting methods are worth a read:

- https://en.wikipedia.org/wiki/Sorting_algorithm
- https://en.wikipedia.org/wiki/Bubble_sort
- https://en.wikipedia.org/wiki/Quicksort
- https://en.wikipedia.org/wiki/Insertion_sort

There is also a web site with animations of several sorting algorithms:

- https://www.toptal.com/developers/sorting-algorithms

A youtube video that gives video and audio animation of a sort:

- https://www.youtube.com/watch?v=kPRA0W1kECg

CHAPTER TWENTYTWO

BIRTHDAY PARADOX

[status: usable-but-incomplete]

22.1 To get started

We start by going around the room and seeing how many people there are. Then we ask everyone to estimate "what's the chance that at least two people have the same birthday?"

We look at the estimates and see how they vary, and ask students why they picked the probability they did.

Then we talk about picking *one person* and asking what's the probability that someone else has their same birthday.

How is that a different question, and would the probability is the same, smaller, or bigger?

22.2 A practical demonstration

Look at the code in Listing **??**:

Listing 22.2.1: Simulate a party with several people and calculate the probability that two of them share a birthday.

```
#! /usr/bin/env python3

"""Simulates the birthday paradox.  You set how many people
are at the party.  The program will assign a random birthday
to each person, and then calculate how mahy duplicate
birthdays are in that group."""

import random

def main():
    n_people = 25
    birthdays = make_birthdays(n_people) # randomly distributed
    print('## Single example with %d people:' % n_people)
    print('## birthdays:', birthdays)
    print('## birthdays_sorted:', sorted(birthdays))
    n_doubles, n_triplets = count_multiples(birthdays)
    print(f'## doubles: {n_doubles}   triplets: {n_triplets}')
    # return                      # stop here for the simplest analysis
    n_tries = 200
```

(continues on next page)

(continued from previous page)

```
    print('## running %d tries' % n_tries)
    for n_people in range(100):
        avg_doubles, avg_triplets = get_average_multiples(n_people, n_tries)
        print(f'n_people: {n_people} -- frac_doubles: {avg_doubles}   frac_triplets:
↪{avg_triplets}')

def make_birthdays(n_people):
    """Generate a birthday for each person, but we do it the easy way: a
    number from 1 to 365, so we don't handle leap years.
    """
    # the list "birthdays" will store the birthday of each person
    birthdays = [0]*n_people
    for person in range(n_people):
        day = random.randint(1, 365) # generate random b-day
        birthdays[person] = day      # store it for that person
    return birthdays

def count_multiples(bdays):
    """Goes through the birthday list and sees if any two people
    have the same birthday.  Returns how many times we find
    duplicate birthdays."""
    n_people = len(bdays)
    count_doubles = 0
    count_triplets = 0
    for person in range(n_people):
        this_bday = bdays[person]
        # we can look at just the "further on" entries in this list
        # since we have already looked for duplicates before this point.
        for other_dude in range(person + 1, n_people):
            other_bday = bdays[other_dude]
            # now see if two people have the same birthday
            if this_bday == other_bday:
                # we have a duplicate!
                count_doubles += 1
                # now look for a third dude; once
                # again I can look beyond this point
                for third_dude in range(other_dude + 1, n_people):
                    third_bday = bdays[third_dude]
                    if third_bday == other_bday:
                        count_triplets += 1 # I have a third!!
    return count_doubles, count_triplets


def get_average_multiples(n_people, n_tries):
    """Estimates an expectation of finding people with the same birthday.
    Returns the probability of two and of three people having the same
    birthday.
    """
    n_with_doubles = 0
    n_with_triplets = 0
    for i in range(n_tries):
        bdays = make_birthdays(n_people)
```

(continues on next page)

(continued from previous page)

```
        n_doubles, n_triplets = count_multiples(bdays)
        if n_doubles >= 1:
            n_with_doubles += 1
        if n_triplets >= 1:
            n_with_triplets += 1
    avg_doubles = (1.0*n_with_doubles) / n_tries
    avg_triplets = (1.0*n_with_triplets) / n_tries
    return avg_doubles, avg_triplets


main()
```

The result of running birthdays.py: this calculates the probability that two people at a party will share the same birthday for party. We put in population sizes of 0 to 50, but you can obviously extend that all the way to 365 or more.

We can plot this output with:

```
python3 birthdays.py > bday_prob.out
```

and plot with:

```
gnuplot
# then at the gnuplot> prompt type:
plot 'bday_prob.out' using 2:7 with linespoints
```

22.3 The theory

The calculation is a bit involved, but it is a very good example of the field of *combinatorics,* and it teaches us a lot about how to count things. Since this example was clearly counterintuitive, the exercise of counting well is a good one.

The simplifying idea is to first calculate the opposite: the probability that *no* two people have the same birthday.

Imagine a sequence of events in which people enter an empty room.

For one person the probability of no duplicate birthdays is 1, which is the same as $365/365 = 1.0$.

Let us say you have 2 people, then the probability that the second person does not have the same birthday as the first is $364/365 \approx 0.99726$.

Add a 3rd person and you have $363/365 \approx 0.99452$ chance that the newcomer's birthday does not match one of the other two.

To get the probability of no matches with 3 people you take the *join* probability of the 3 situations: 1.0 when the first person enters the room, times the probabilities of no matches with each successive entry into the room:

$$\begin{aligned} P_{\text{no\_dup}}(n) &= \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \approx 0.991795 \\ P_{\text{duplicates}}(n) &= 1 - P(\text{no\_dup}) \approx 1 - 0.991795 \approx 0.00820417 \end{aligned}$$

The general formula after n people have entered the room is:

$$P_{\text{no\_dup}}(n) = \frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{1}{365} = \frac{365 \times (365-1) \times \cdots \times (365-n+1)}{365^n}$$

Using the definition of factorial, we see that the numerator is:

$$\begin{aligned} \text{numerator} =& 365 \times 364 \times 363 \times ... \times (365-n+1) = \frac{365!}{(365-n)!} \\ =& n! \times \binom{365}{n} \end{aligned}$$

where we used the mathematical "choose" notation:

$$\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1} = \frac{n!}{k!(n-k)!}$$

So our formula is:

$$\begin{aligned} P_{\text{no\_dup}}(n) &= \frac{n! \times \binom{365}{n}}{365^n} \\ P_{\text{duplicates}}(n) &= 1 - \frac{n! \times \binom{365}{n}}{365^n} \end{aligned}$$

Do we believe this? Well, let us compare it to our monte carlo calculation for the tipping point of $n = 23$:

$$P_{\text{duplicates}}(23) = 1 - \frac{365! \times \binom{365}{23}}{365^{23}} \approx 1 - 0.492703 \approx 0.507297$$

22.4 Take-home

What we have learned:

- Simulate a situation (in this case people sharing birthdays).
- Calculate the probability of an event with a random component. We do this by running the event many times and averaging the outcome.
- Jargon: you could think of this as a very simple example of a "monte carlo" simulation.

CHAPTER

TWENTYTHREE

GRAPHICAL USER INTERFACES

[status: not-yet-written]

Prerequisites

- The 10-hour “serious programming” course.
- A GNU/Linux system with python3 and python3-tk installed (on an Ubuntu 16.04 system this can be done with `sudo apt install python3-tk`)

23.1 A chat about sources of input in a GUI

The programs we first write with text-based user interfaces, like the tic-tac-toe program in the basic course, have a very simple flow:

1. You get input from the keyboard (and *only* from the keyboard).
2. You do some processing based on that input.
3. You write some output to the terminal.

Later, in Section **??**, we learned to write programs which read data from a file. Still, even in this kind of program you are always only reading input from one place at a time.

At this point I talk to the class about how there are two types of programs that do things quite differently: **network** programs and **GUI** programs.

In network programs you can have connections open to several different hosts, and you might be reading input from several of them at once.

In A GUI program you can have input from the keyboard, you can also have mouse movement and mouse clicks. The program also has to track other events that can affect its behavior, like the movement of a window so that a different portion of it is exposed.

GUI programs often use what’s called an *event loop*: you set up your programs layout with windows and widgets and then go in to an infinite loop where you call a function that checks if any events (mouse, keyboard, . . .) have occurred.

In discussing this I would then to go the whiteboard and draw an example of a simple GUI with a couple of buttons. With that up I would discuss *what happens when the user clicks on something?*

This leads to the discussion of a *callback* or *callbacks function*.

23.2 Widgets and widget sets

Discuss what are widgets? Little self-contained user interface bits (like buttons and text entry fields and canvases) which you can place in your program's graphical interface.

As is often the case in the free software world, there is a dizzying array of different *widget sets* available for programming in Python. Tkinter, wxPython, PyQt, PyGTK, Kivy, . . .

The most basic widget set, Tkinter, is included as a standard part of Python. It allows you to develop reasonable comprehensive graphical interfaces, so I will use that as our starting point.

23.3 The simplest programs

23.3.1 The programs

Simple single OK button program.

Listing 23.3.1: ok-simplest.py - program with an OK button.

```python
#! /usr/bin/env python3

import tkinter as tk

root = tk.Tk()
okButton = tk.Button(root, text='OK')
okButton.pack()

root.mainloop()
```

Sure, but that does nothing. Let's make it do something when you press the button:

Listing 23.3.2: ok-callback.py - program with an OK button that does something.

```python
#! /usr/bin/env python3

import tkinter as tk

def printOK():
    print('OK!!')

def main():
    root = tk.Tk()
    okButton = tk.Button(root, text='OK', command=printOK)
    okButton.pack()

    root.mainloop()

main()
```

Now let's make it able to quit with a "Quit" button:

Listing 23.3.3: ok-callback-quit.py - program with an OK button that does something simple, and a Quit button that exits.

```
#! /usr/bin/env python3

import tkinter as tk

def printOK():
    print('OK!!')

def main():
    root = tk.Tk()
    okButton = tk.Button(root, text='OK', command=printOK)
    okButton.pack()
    quitButton = tk.Button(root, text='Quit', command=root.destroy)
    quitButton.pack()

    root.mainloop()

main()
```

Observing this simple program raises a couple of questions about things we saw in it:

23.3.2 Packers: more than just one button

One concept in GUI programs is *geometry management*. How does the program lay out all the widgets?

A programmer could say: put this widget at coordinages (12, 74), and this other one at coordinates (740, 210), and so forth. This would be *terrible* style. It would do the wrong thing if the window gets resized, and it would become impossible to maintain when there are more widgets.

Widget systems introduce the idea of "geometry management" to deal with this. The calls to `button.pack()` that you saw in `ok-callback-quit.py` are an example. We told the widgets to pack themselves inside their parent window, and much was taken care of automatically by doing that. We did not, for example, have to specify the position of the buttons in the window. If you resize you will notice that the buttons are kept in somewhat reasonable positions.

As we go on our tutorial tour of widgets let us pay attention to what happens with the packing of widgets inside windows.

23.3.3 A tour of widgets

We only saw button widgets, but this is a chance to point out what other widgets there are. It's hard to get an exhaustive list since one can write custom widgets, but here are some to mention:

- button
- canvas
- checkbutton
- combobox
- entry
- frame
- label

- labelframe
- listbox
- menu
- menubutton
- message
- notebook
- tk_optionMenu
- panedwindow
- progressbar
- radiobutton
- scale
- scrollbar
- separator
- sizegrip
- spinbox
- text
- treeview

23.4 Following a tutorial

We will now follow this tutorial:

https://www.tutorialspoint.com/python3/python_gui_programming.htm

This tutorial has links to most of the basic Tkinter widgets, with examples of how to use each one. In the course I have the students bring them up one by one, pasting them in to the python3 interpreter to see them at work.

The one that might be most interesting is the Scale widget: it shows three interacting widgets where the slider sets a value, the button then reads that value and causes a label to be updated. I would go through that example in greater detail.

23.5 Cellular automata on a canvas

Students might want to read this before going through this chapter:

https://en.wikipedia.org/wiki/Elementary_cellular_automaton

23.5.1 A simply drawing of the CA

We have an example of a cellular automaton elsewhere in the book (Section **??**). We can use the routines in that program to compute the cellular automaton, and just add the graphical portion in this program.

Download the simple_ca.py program from Section **??** and save it in a file called `simple_ca.py`.

Then take a look at the `draw_ca.py` program in Listing **??** and try running it.

Listing 23.5.1: draw_ca.py - draws a cellular automaton graphically.

```
#! /usr/bin/env python3

"""draw a cellular automaton"""

import time
import math
import sys

sys.path.append('../emergent-behavior')
from simple_ca import *

## we use the tkinter widget set; this seems to come automatically
## with python3 on ubuntu 16.04, but on some systems one might need to
## install a package with a name like python3-tk
from tkinter import *

def main():
    ## how many steps and cells for our CA
    n_steps = 200
    n_cells = 200
    ## we will make each cell be 4x4 pixels
    canvas_width = 4*n_cells
    canvas_height = 4*n_steps

    ## prepare a basic canvas
    root = Tk()
    ca_canvas = Canvas(root,
                       width=canvas_width,
                       height=canvas_height)
    ca_canvas.pack() # boiler-plate: we always call pack() on tk windows

    # row = set_first_row_random(n_cells)
    # row = set_first_row_specific_points(n_cells, [40])
    row = set_first_row_specific_points(n_cells, [12, 40, 51, 52, 60, 110, 111,
                                                  160, 161, 162, 163, 164, 165,
                                                  166, 167, 168, 169, 170, 171, 177])
    # row = set_first_row_specific_points(n_cells, list(range(int(n_cells/2), n_cells)))
    # row = set_first_row_specific_points(n_cells, [12, 13, 50, 51, 70, 71, 99, 100])

    ## now set the rule
    rule = '01101000'           # the basic rule
    # rule = '00011110'          # the famous rule 30
    # rule = '01101110'          # the famous rule 110
```

(continues on next page)

(continued from previous page)

```
    draw_row(ca_canvas, 0, row)
    for i in range(1, n_steps):
        row = take_step(rule, row)
        draw_row(ca_canvas, i, row)
    mainloop()

def draw_row(w, pos, row):
    color_map = ['black', 'white', 'red', 'green', 'blue', 'yellow']
    for i, cell in enumerate(row):
        color = color_map[cell % len(color_map)]
        w.create_rectangle(4*i, 4*pos, 4*i+4, 4*pos+4, fill=color)
        w.update()
        ## update the canvas
        # if i % 10 == 0:
        #     w.update()
    w.update()


if __name__ == '__main__':
    main()
```

Change the initial conditions, the size of the automaton, experiment with it.

Exercise 23.1 Modify draw_ca.py to have a *first row editor*: a small canvas which allows you to modify the cells in the first row so that you can run any configuration you choose.

Exercise 23.2 Modify draw_ca.py to have a *rule editor*: a small canvas which allows you to modify the rule for the cellular automaton evolution. You should also have widgets to allow a different number of cells, a different number of rows, and more or less states or neighbors.

Exercise 23.3 You might have noticed that the drawing of the canvas gets slower and slower as you have more and more squares. You can get around this by commenting out the `w.update()` call so that the cellular automaton only gets drawn at the end of the run, but then you miss out on seeing the progression of the cellular automaton. Modify `draw_ca.py` to be more efficient in drawing the canvas. You can use the information at this stack overflow link as a guide:

https://stackoverflow.com/questions/10515720/tkinter-canvas-updating-speed-reduces-during-the-course-of-a-program

23.5.2 Adding controls to the program

Then save `draw_ca_with_controls.py` and study it and run it. At this time the program is not yet too clean (FIXME: update this text when I update the program), but we can discuss the new mechanisms that appear in this program:

- The canvas does not just run automatically: we control it with the buttons.
- We use the canvas's `after()` function to make the canvas draw "in the background" while the controls are still active. This means we can pause, for example.

Exercise 23.4 Introduce a "first row editor" widget between the controls and the canvas. This recognizes mouse clicks and lets you set the initial cell values so you don't have to set them with code in the program.

Exercise 23.5 introduce a "rule editor" widget, possibly on the same row as the control widgets. You could start by just taking a number between 0 and 255. Them move on to 8 squares or checkboxes, where you would click on them to activate the binary cells that end up in the rule string. But the coolest might be to make a widget that shows the 3 cells and their child, with the 8 possible 3-cell configurations, and picking if they turn in to 1 or 0.

23.6 Conway's game of life

Goal: create a canvas which allows you to click to select initial cell values. Then kick off Conway's game of life rule.

I don't have much explanatory text yet, but for now let's discuss the program we have.

Download the `conway_life.py` program from Section **??** and save it in a file called `conway_life.py`.

Then save `draw_conway_life.py`.

We study the programs together and see how the GUI version works, and then run it.

23.7 Tic-tac-toe with buttons

Listing 23.7.1: ttt-gui.py - GUI for the tic-tac-toe program we wrote in the basic Python course.

```
#! /usr/bin/env python3

import sys
import os
import tkinter as tk

import tic_tac_toe as ttt

class TTTGui(tk.Frame):
    def __init__(self, parent, _xHuman, _yHuman):
        """Draw the initial board"""
        tk.Frame.__init__(self, parent)
        ## set up some constant variables that last across games
        self.imBlank = tk.PhotoImage(file='green_background.png')
        self.imX = tk.PhotoImage(file='green_back_X.png')
        self.imO = tk.PhotoImage(file='green_back_O.png')
        self.marker2image = {' ': self.imBlank,
                             'x': self.imX,
                             'o': self.imO}
        ## zero out
        self.ResetGame(_xHuman=_xHuman, _yHuman=_yHuman)
        ngB = tk.Button(self, text='New game', command=lambda:
                        self.ResetGame(_xHuman=_xHuman, _yHuman=_yHuman))
        ngB.grid(row=4, column=0)
        qB = tk.Button(self, text='Quit', command=self.Quit)
        qB.grid(row=4, column=1)
        self.grid(sticky=tk.N+tk.S+tk.E+tk.W)

    def ResetGame(self, _xHuman=True, _yHuman=False):
        ## rest the game state variables
        self.bd = ttt.new_board()
        self.gameState = {'xHuman': _xHuman,
                          'yHuman': _yHuman,
                          'toMove': 'x',
                          'winner': ttt.find_winner(self.bd)}
        ## set up the board buttons from scratch
```

(continues on next page)

(continued from previous page)

```
        self.buttons = [[None, None, None],
                        [None, None, None],
                        [None, None, None]]
        for row in range(3):
            for col in range(3):
                self.buttons[row][col] = tk.Button(self, image=self.imBlank)
                self.buttons[row][col].grid(row=row, column=col)
                ## we bind the button to the PlaceMarker() method,
                ## making sure to pass it the row and column
                self.buttons[row][col].bind('<Button-1>',
                                            lambda event, row=row, col=col:
                                            self.PlaceMarker(row, col))
        self.UpdateBoard()

    def PlaceMarker(self, row, col):
        if self.gameState['winner'] != ' ':
            print('GAME_IS_WON, not placing marker')
            return
        if ttt.board_is_full(self.bd):
            print('BOARD_IS_FULL, not placing marker')
            return
        if self.bd[row][col] == ' ':
            print('PLACE: %d, %d' % (row, col))
            ttt.set_cell(self.bd, row, col, self.gameState['toMove'])
            self.HandlePossibleEnd()
            self.UpdateBoard()
            self.gameState['toMove'] \
                = ttt.next_marker(self.gameState['toMove'])
            if self.gameState['winner'] == ' ':
                if (not self.gameState['xHuman']
                    or not self.gameState['yHuman']):
                    if not ttt.board_is_full(self.bd):
                        self.TriggerComputerMove()
        else:
            print('ILLEGAL: %d, %d' % (row, col))
            pass                # invalid move
        self.HandlePossibleEnd()

    def UpdateBoard(self):
        for row in range(3):
            for col in range(3):
                image = self.marker2image[self.bd[row][col]]
                self.buttons[row][col].configure(image=image)
        self.HandlePossibleEnd()

    def TriggerComputerMove(self):
        ttt.play_computer_opportunistic(self.bd, self.gameState['toMove'])
        self.gameState['toMove'] = ttt.next_marker(self.gameState['toMove'])
        self.UpdateBoard()
        self.HandlePossibleEnd()

    def HandlePossibleEnd(self):
```

(continues on next page)

(continued from previous page)

```
        self.gameState['winner'] = ttt.find_winner(self.bd)
        print('WINNER: <%s>' % self.gameState['winner'])
        if ttt.board_is_full(self.bd):
            print('I *should* put up some info')
            print('WINNER: <%s>' % self.gameState['winner'])

    def Quit(self):
        self.master.destroy()

def main():
    app = TTTGui(tk.Tk(), True, False)
    app.mainloop()

if __name__ == '__main__':
    main()
```

You can use that GUI program with the underlying text-based program we wrote in the basic Python course: `tic_tac_toe.py`

23.8 Other resources

https://tkdocs.com/tutorial/ (does many languages side-by-side)

http://zetcode.com/gui/tkinter/ (object oriented; this is too soon to use OOP in this course)

https://likegeeks.com/python-gui-examples-tkinter-tutorial/ (maybe good)

https://dzone.com/articles/python-gui-examples-tkinter-tutorial-like-geeks (maybe good)

https://www.python-course.eu/tkinter_labels.php (maybe good because it has modern preferences like "import tkinter as tk", but maybe too lengthy in the early examples)

https://www.tutorialspoint.com/python3/python_gui_programming.htm

CHAPTER

TWENTYFOUR

DRAWING ON A CANVAS

[status: content-mostly-written]

Prerequisites

- The 10-hour "serious programming" course.
- A GNU/Linux system with python3 and python3-tk installed (on an Ubuntu 16.04 system this can be done with `sudo apt install python3-tk`)

24.1 Simplest canvas

Drawing is usually done with the help of a graphical toolkit library. This library usually supports *widgets* and allows them to be arranged in a window on the scren. The widget for drawing is called a *canvas* and in Listing **??** is a simple example using Python's `tkinter` widget set.

Listing 24.1.1: Program which draws two circles and a line.

```
#! /usr/bin/env python3

"""simple canvas with two discs and a line between them
"""

## we use the tkinter widget set; this seems to come automatically
## with python3 on ubuntu 16.04, but on some systems one might need to
## install a package with a name like python3-tk
from tkinter import *

canvas_width = 640
canvas_height = 480

def main():
    ## prepare a basic canvas
    root = Tk()
    w = Canvas(root,
               width=canvas_width,
               height=canvas_height)
    w.pack()       # boiler-plate: we always call pack() on tk windows
    w.create_oval(210, 230, 230, 250, fill="yellow")
    w.create_oval(410, 230, 430, 250, fill="blue")
```

(continues on next page)

(continued from previous page)

```
    w.create_line(220, 240, 420, 240)
    mainloop()

main()
```

Things to notice about this program:

- It uses the library `tkinter` which provides calls to create a window and draw things in it.
- The calls `root = Tk()` and `w.pack()` and `mainloop` are boiler-plate: almost all programs that use the tkinter toolkit will use them.
- `w = Canvas(...)` creates a canvas with the given width and height. `w.pack()` places that canvas into the window we are using.
- After the drawing calls `create_oval()` and `create_line()` we call `mainloop()`. This enters an *event loop* in which the widget set is waiting for events, such as the click of a button. We have not created any buttons, so `mainloop()` will just hang until we interrupt the progrma.

24.2 Simplest animation

Animation can be done simply by drawing onto a canvas, then changing the drawing after a small amount of time and refreshing the canvas.

In listing Listing **??**

Listing 24.2.1: Program which animates a circle whose radius is growing and shrinking.

```
#! /usr/bin/env python3

"""simple animation of a growing/contracting disc
"""

import time
import math

## we use the tkinter widget set; this seems to come automatically
## with python3 on ubuntu 16.04, but on some systems one might need to
## install a package with a name like python3-tk
from tkinter import *

canvas_width = 640
canvas_height = 480

def main():
    ## prepare a basic canvas
    root = Tk()
    w = Canvas(root,
               width=canvas_width,
               height=canvas_height)
    w.pack()       # boiler-plate: we always call pack() on tk windows
    for i in range(24*180):     # 3 minutes if it's 24 frames/sec
```

(continues on next page)

(continued from previous page)

```
        radius = 140 + 50*math.sin(i/24.0) # oscillating radius
        center = (canvas_width/2, canvas_height/2)
        color = 'blue'
        ## clear the canvas and then draw a new disc
        w.delete('all')
        w.create_oval(center[0]-radius, center[1]-radius,
                      center[0]+radius, center[1]+radius,
                      fill=color)
        ## update the canvas
        w.update()
        time.sleep(1.0/24.0)    # 24 frames per second
    mainloop()

main()
```

Note that this kind of animation is not ideal: it works well, but it does not allow the program to respond to any user input or other events. For us this is OK because we are purely showing the animation, and we are not setting up any buttons or other parts to the program, but the `tkinter` library offers a more appropriate approach to doing animations with a method called `after()` which allows you to call your drawing routines and handle user interface events at the same time.

24.2.1 Exercises

Exercise 24.1 Modify the tirial animation program in Listing **??** to write the current at the bottom of the canvas. A web search for "tkinter write string on canvas" might help.

Exercise 24.2 Study how colors can be represented as combinations of *red*, *green* and *blue* (RGB) values. Read the introductory part of the Wikipedia article on the subject at https://en.wikipedia.org/wiki/RGB_color_model and then use the program *gpick* to examine how the various screen pixel colors can be represented as red, green and blue. Explore the program in detail, looking at how you can specify the RGB values and see what the result is, but you can also pick them from a color wheel, and you also use the *screen picker* FIXME. . .

Exercise 24.3 Modify the trivial animation program in Listing **??** to cycle through the colors as well as the radius. Aim for a psychadelic effect. Note that the `fill=` field in `create_oval()` can be a descriptive color name (like the `"blue"` that we used) but it can also specify the RGB (red, green, blue) values that form the color. A web search for "tkinter colors" might help.

CHAPTER

TWENTYFIVE

THE TRAVELING SALESMAN

[status: content-mostly-written]

Applications of *optimization*

Optimization in practice

The traveling salesman problem (TSP) is one of the best known examples of *optimization*, which is a very important field. Its solution is relevant to many large scale logistical efforts, such as the most efficient path for school buses in a city, or airline routing, or freight transportation. To show how important this is regarded as, in 1962, the giant corporation Proctor and Gample ran a contest to solve an instance of TSP with 33 U.S. cities. The prize was $75000 in 2013 dollars. In the world of theoretical computer science, the TSP spawned the development of many of the most important advances in computer science. Although one might not go as far as to quote Mandos in The Silmarillion and say that the fate of Earth is linked to solutions to the TSP, it is hard to overstate the importance of being able to find good solutions to these types of problems.

Motivation

One of the most important things done by computer software is to find *approximations* to mathematical problems that are too complex to carry out exactly. We will examine one of these problems, the Traveling Salesman Problem (TSP), which is easy to formulate and has many practical applications.

Prerequisites

- The 10 hour "serious programming" course.
- A GNU/Linux system with python3 and python3-tk installed (on an Ubuntu 16.04 system this can be done with `sudo apt install python3-tk`)
- The mini-course on drawing on canvases, in Section **??**

Plan

We will start by formulating what the problem looks like, then we will discuss the general idea of optimization, after which we will write programs in Python that find an easy (but not very good) solution with the "greedy" algorithm. Finally we will write a program which uses the "hill climbing" algorithm to find an approximate solution.

Throughout these examples we will use the simple drawing techniques we learned in Section **??** to show rather cute animations of our paths through the cities.

And this video snippet, which discusses Buddy Holly's "Winter Dance Tour", mentions the poorly chosen route through the midwest: https://youtu.be/NFdWrwbxNms?t=214 – the cities were, in this order: Milwaukee WI, Kenosha Wi, Mankato MN, Eau Claire Wi, Montevideo MN, St. Paul MN, Davenport IA, Fort Dodge IA, Duluth MN, Green Bay WI, Clear Lake IA, Moorhead MN, Sioux City IA, Des Moines IA, Cedar Rapids IA, Spring Valley IL, Chicago IL, Waterloo IA, Dubuque IA, Louisville KY, Canton OH, Youngstown OH, Peoria IL, Springfield IL.

https://commons.wikimedia.org/wiki/File:Winter_Dance_Party_Tour_Schedule,_1959.svg

25.1 Cities and path lengths

Our cities are a list of (x, y) coordinates within the canvas size. A future improvement would be to use realistic latitude and longitude values and fit those into a canvas.

The length of the path is the sum of the space between each successive pair of cities, and the distance between two cities can be calculated with the Pythagoras theorem (IMPROVEME: a simple picture would be nice here). If the cities have coordinates (x_1, y_1) and (x_2, y_2), then the x and y distances between them are $(x_2 - x_1)$ and $(y_2 - y_1)$, and the distance "as the crow flies" is given by the Pythagorean formula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

If we were to represent the coordinates of a city in Python with a simple pair then we might represent two cities like this:

```
c1 = (14, 182)
c2 = (71, 50)
```

We might then write a function to calculate the distance between them like this:

```
import math
## ...
def distance(city1, city2):
    x1 = city1[0]    ## coordinates of the first city
    y1 = city1[1]
    x2 = city2[0]    ## coordinates of the second city
    y2 = city2[1]
    r = math.sqrt((x2-x1)**2 + (y2-y1)**2)
    return r
```

Once we have this function we could calculate the distance like this:

```
c1 = (14, 182)
c2 = (71, 50)
d = distance(c1, c2)
print('The distance between the cities (%d, %d) and (%d, %d) is %g'
      % (c1 + c2 + (d,)))
```

The output would look like this:

```
The distance between the cities (14, 182) and (71, 50) is 143.781
```

To calculate the distance of the *entire path* we would add together all the distances from the first city to the second, from the second to the third, and so on until the last one. Finally we would add the "return home" path length from the last city back to the first.

25.2 Solving the Traveling Salesman Problem

What does it mean to "solve" the TSP? It means to find the *shortest* path between the cities. We will see that for larger sizes this is not practically possible, but we will try to do better than a random path, and to take some steps to further improve our solutions.

25.2.1 A digression on optimization

Show pictures of a search in 1d and 2d, possibly with simple 1-d and 2-d gaussian pictures.

The optimization task we are looking at (the traveling salesman problem) is an interesting one, but let us digress to look at some simpler optimization tasks for which we can make interesting plots.

Let us examine the following *gaussian* function $e^{(-(x-3)^2)}$ which you might remember from Section **??**:

```
$ gnuplot
gnuplot> set samples 400
gnuplot> plot exp(-(x-3)**2)
gnuplot> set terminal svg
gnuplot> set output 'single-hill-1d.svg'
gnuplot> replot
gnuplot> quit
```

Note that you could have similar function that points downward:

```
$ gnuplot
gnuplot> plot -exp(-(x-3)**2)
```

In the first case our function has *maximum*, and in the second case it has a *minimum*. The word *optimum* can refer to one or the other.

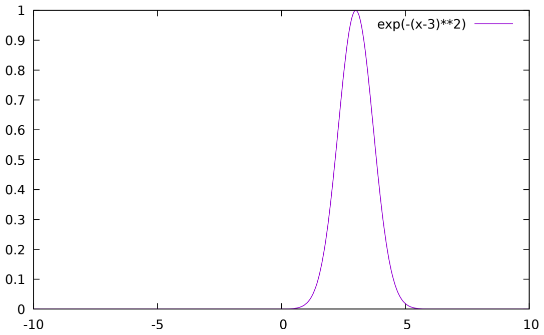


Figure 25.2.1: An example of a one dimensional hill-shaped function. Note the single optimum: the top of the hill for $x = 3$

Now try a two dimensional function with a clear single hill and single optimum:

```
$ gnuplot
gnuplot> set pm3d
gnuplot> set samples 150
gnuplot> set isosamples 60
gnuplot> set hidden3d
gnuplot> splot exp(-(x-1)**2 - (y-0.5)**2)
gnuplot> set terminal svg
gnuplot> set output 'single-hill-2d.svg'
gnuplot> replot
gnuplot> quit
```

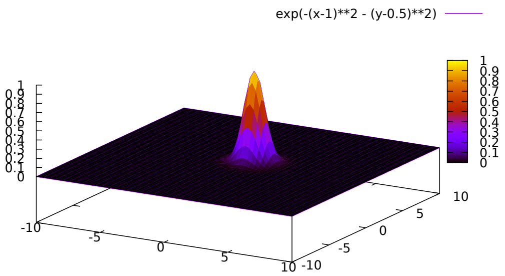


Figure 25.2.2: An example of a two dimensional hill-shaped function. Note the single optimum: the top of the hill for $(x, y) = (1, 0.5)$

In Figure **??** and Figure **??** we see that some optimization problems are as simple as climbing to the top of a single nearby hill. There are no other hills to confuse us and it should be simple to find the optimal value of x (in one dimension) or of x and y (in two dimensions).

These figures give a clear idea of what we are looking for in optimization when we optimize a function of x or of x and y. The picture is not so pretty when we try to optimize something more complicated: there is no such visualization of the distance in the traveling salesman problem, since city routes cannot be plotted as an x axis or as x and y axes. When we get to it, we will visualize the improvement in the TSP by showing an animation of the path through the cities.

25.2.2 Generating and visualizing lists of cities

Examine and type in the program in Listing **??**

Listing 25.2.1: Program which draws a simple set of random cities on a canvas, in file `cities_simple.py`.

```
#! /usr/bin/env python3

"""This program demonstrates basic generation of a canvas: it makes a
list of random city coordinates, then draws them (with paths) on the
canvas.
"""

import random
```

(continues on next page)

(continued from previous page)

```
## we use the tkinter widget set; this seems to come automatically
## with python3 on ubuntu 16.04, but on some systems one might need to
## install a package with a name like python3-tk
from tkinter import *

canvas_width = 640
canvas_height = 480
n_cities = 25

def main():
    ## prepare a basic canvas
    root = Tk()
    w = Canvas(root,
               width=canvas_width,
               height=canvas_height)
    w.pack()       # boiler-plate: we always call pack() on tk windows
    city_list = make_random_cities(0, canvas_width-1, 0, canvas_height-1, n_cities)
    draw_city_path(w, city_list)
    mainloop()

def draw_city_path(w, city_list):
    """draws lines between the cities"""
    for city in city_list:
        draw_city(w, city[0], city[1])
    draw_city(w, city_list[0][0], city_list[0][1], color='green', name='Home')
    ## now draw lines between them
    for i in range(len(city_list)-1):
        w.create_line(city_list[i][0], city_list[i][1],
                      city_list[i+1][0], city_list[i+1][1])
    ## now draw a line that goes from the last city back to our home
    w.create_line(city_list[-1][0], city_list[-1][1],
                  city_list[0][0], city_list[0][1])


def make_random_cities(xmin, xmax, ymin, ymax, n_cities):
    """returns a list of randomly placed cities in the given rectangle"""
    city_list = []
    for i in range(n_cities):
        x = random.randint(xmin, xmax)
        y = random.randint(ymin, ymax)
        city_list.append((x, y))
    return city_list

def draw_city(w, x, y, color='yellow', name=None):
    """draws a city; if a name is given also writes the name of it"""
    w.create_oval(x-5, y-5, x+5, y+5, fill=color)
    ## if a name was given, write in the name
    if name:
        w.create_text(x, y+10, text=name)

main()
```

Exercise 25.1 Modify `cities_simple.py` to print some information about the path of these cities at the bottom of the canvas. (An example can be downloaded in `cities_simple_with_info.py`.)

25.2.3 Animating the drawing of cities

We can modify the program in Listing **??** quite straightforwardly to modify some information about the list of cities and animate the city drawing to visualize those changes.

You can download this program `cities_animated.py` and try running it to see how it works.

Exercise 25.2 Modify `cities_animated.py` to keep the *starting* point of the search.

Exercise 25.3 Write a function which calculates the distance between two cities, then write a function which calculates the total length of a list of cities.

Exercise 25.4 Modify `cities_simple.py` and `cities_animated.py` to print the total length of the path and other interesting information at the bottom of the screen. (An example can be downloaded in this file: `cities_animated_with_info.py`.)

25.3 Improvements to the route

25.3.1 Before you start

- https://www.youtube.com/watch?v=xi5dWND499g (british fellow gives a hands-on demonstration of TSP with a map, pushpins, and a length of string)
- https://www.youtube.com/watch?v=SC5CX8drAtU (attractive and partially annotated visualization of greedy, local-search/hill-climbing and simulated annealing algorithms)

25.3.2 Impossibile to compute the optimal solution

People will tell you that the traveling salesman problem cannot be solved by "direct attack" because there are too many possible paths for a computer to explore them all.

Let us look in to this with pencil and paper in hand and see if we agree.

Draw three cities on a sheet of paper, pick a first city. Draw all paths that start with that first city and eventually get you back home to it. There should be 2 different paths.

Now do the same with 4 cities. There should be 6 paths.

Try to do the same with 5 cities. There should be 24 possible paths.

In general the equations is:

$$n_{paths}(n_{cities}) = (n_{cities} - 1)! = (n_{cities} - 1) \times (n_{cities} - 2) \times \ldots \times 3 \times 2 \times 1$$

where the -1 in $n_{cities} - 1$ comes in because at the end of the run you always return to the given home city.

Now try to argue with yourself and with your partners to convince everyone that this factorial formula works for larger

n_{cities}. And how rapidly does $(n_{cities} - 1)!$ grow? Remember from Section **??** that it grows at an amazingly fast rate:

$$
\begin{aligned}
2! &= 2 \times 1 = 2 \\
3! &= 3 \times 2 \times 1 = 6 \\
4! &= 4 \times 3 \times 2 \times 1 = 24 \\
5! &= 5 \times ... \times 2 \times 1 = 120 \\
6! &= 6 \times ... \times 2 \times 1 = 720 \\
7! &= 7 \times ... \times 2 \times 1 = 5040 \\
8! &= 8 \times ... \times 2 \times 1 = 40320 \\
9! &= 9 \times ... \times 2 \times 1 = 362880 \\
10! &= 10 \times ... \times 2 \times 1 = 3628800 \\
&... \\
20! &= 10 \times ... \times 2 \times 1 = 2432902008176640000 \\
&... \\
32! &= 32 \times 31 \times ... \times 2 \times 1 = 263130836933693530167218012160000000
\end{aligned}
$$

So 32! is a number with 35 digits, approximately 2.63131×10^{35}, so clearly we cannot hope for a computer to look at all possible paths. Thus we look for approximate ways of doing it.

25.3.3 Greedy algorithm

One algorithm that comes to mind is to always travel to the city that is closest to you and that you have *not* yet visited. This is not the best way, but it is usually not the worst way either.

Exercise 25.1 With pen and paper draw short routes (4 or 5 cities) and solve them with the greedy algorithm. Discuss with your partners if this is the optimal solution or not.

Exercise 25.2 You can download this program `tsp_solution_greedy.py` and try running it to see how it works. Change the number of cities to be quite small and quite big.

Exercise 25.3 Discuss with your partners what the term *deterministic algorithm* might mean, and whether the greedy algorithm is deterministic.

Exercise 25.4 With your partners try to figure out (with pen and paper) how many times the greedy program calculates the distance between two cities. This is called the *computational complexity* of the algorithm. This number should be a function of `n_cities`.

Exercise 25.5 Modify the program to *count* how many times it calculates the distance between two cities. Then run it with a variety of different values for `n_cities`.

Exercise 25.6 Try to come up with a layout of cities in which the greedy algorithm performs very poorly. After you have tried conjuring a set of your own, you may look at [Bir15] for some examples.

Exercise 25.7 Try swapping triplets of cities instead of pairs and see if it does better. A couple of things to consider as you try this:

- Triplets might make it harder to reach an optimum because sometimes the best improvement in hill-climbing might come from a simple 2-point swap. So you might need to explore sometimes swapping 2, sometimes 3, maybe even more.
- To compare 2-swaps or 3-swaps or even more, you might want to set the *random number seed* to a fixed value at the start so that you can reproduce the same layout of cities. In python you can do this with `random.srandom(1234)` near the start of the program.

Exercise 25.8 Learn about the “basin hopping” algorithm provided by the scipy python library, and see if it can be used to solve the traveling salesman problem. Write a program which uses this method and run it alongside the program we wrote which uses the hill climbing algorithm.

25.3.4 A digression on hill climbing

Before you start

- Watch this video: https://www.youtube.com/watch?v=kOFBnKDGtJM (Georgia Tech course on machine learning – hill-climbing section. Pedagogically quite understandable.)
- Re-examine some of the plots in our tour of functions in Section **??**, specifically in Section **??**.

25.3.5 Hill climbing for the traveling salesman problem

So what is hill-climbing? It is a simple algorithm (expressed for when we seek a *maximum*):

1. Start at a certain location (random, or you can pick it if you know something about the landscape).
2. Try taking a step in a random direction.
3. Calculate your function. If it is bigger, take that step; if it is smaller, go back to where you were.

If we try to climb up the hills shown in above in Figure **??** and Figure **??** we find it quite straightforward: there is a single peak and we will find it easily.

Now look at different type of hilly functions that we might need to optimize. For example in one dimension we might look at the function $\sin(6 * x) * e^{(x-3)^2}$

We can generate a plot with:

```
gnuplot> set samples 400
gnuplot> plot[-4:] sin(6*x) * exp(-(x-3)**2)
gnuplot> set terminal svg
gnuplot> set output 'multiple-hills-1d.svg'
gnuplot> replot
gnuplot> quit
```

Now look at a 2-dimensional situation where the function could be $e^{-(x^2+y^2)/7} \times \cos(2.5 \times \sqrt{x^2+y^2})$

```
set pm3d
set samples 250
set isosamples 80
set hidden3d
splot exp(-(x**2 + y**2)/7.0) * cos(2.5*sqrt(x*x + y*y))
set terminal svg
set output 'multiple-hills-2d.svg'
replot
```

In the functions show in Figure **??** and Figure **??** we will run in to the problem of climbing to a *local* maximum. This is similar to when you go hiking and it’s foggy and you think you reached the peak of the main mountain, but then the sky clears and you realize that you got stuck on one of the lesser peaks.

Another thing to say about the hill-climbing algorithm is that it is a *stochastic* approach. This means that some of the decisions made by the algorithm involve generating random numbers.

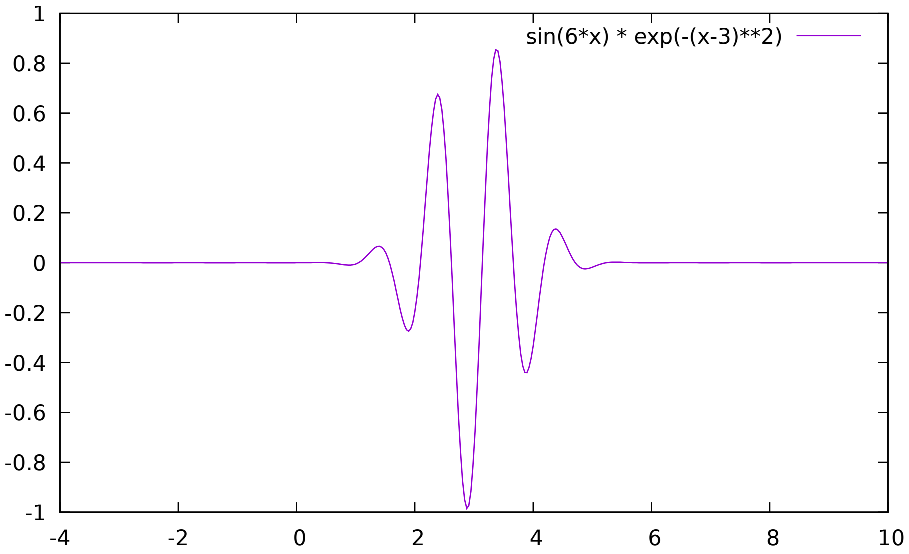


Figure 25.3.1: A one dimensional example of multiple hills with different heights. The *global* maximum is for a value of x near 3.3.

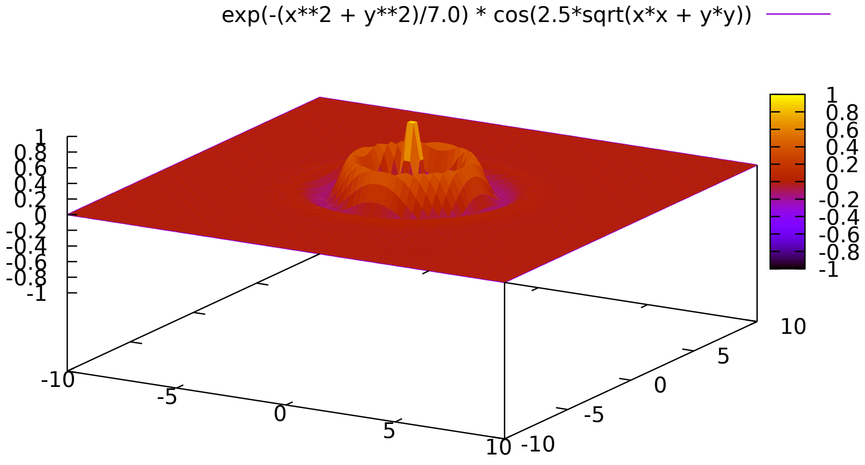


Figure 25.3.2: A two dimensional example of multiple maxima. The *global* maximum is near $(x, y) = (0, 0)$.

You can download and run the program `tsp_solution_hillclimbing.py`. Examine the source code, then run it a few times.

You will see that it starts out with a random and rather chaotic path through the cities. Then it will attempt to swap two cities picked randomly in the path and that swap will stick when the new path is shorter than the previous one.

Exercises

- Use the terminal output from `tsp_solution_hillclimbing.py` to make a plot of the path distance as a function of how many steps have been taken. You could do so by redirecting the output into a file called `hill.out` and using gnuplot to plot columns 2 and 3.
- Discuss with your class how far apart are the cities get swapped successfully. Are they near each other on the list? Are they far apart? Are they near each other in distance? Does that change as we get further in the run?
- Add more information to the `print()` statement to show how far apart the swapped cities are in the list and in distance. Plot how those distances change in the course of a search.

25.3.6 Further study

These lectures on youtube are at a more advanced level than what we do here.

- https://www.youtube.com/watch?v=boTeFM-CVFw&t=45s (Delightful discussion of hill climbing.)
- https://www.youtube.com/watch?v=j1H3jAAGlEA&t=998s (Extensive discussion of search.)
- https://www.youtube.com/watch?v=eczhFRfo3mI (Extensive theoretical discussion.)
- https://www.youtube.com/watch?v=K7vc60jn1KU (Georgia Tech course on machine learning – simulated annealing section. Nice pedagogical introduction.)
- http://aperiodical.com/2015/03/apiological-part-3/ (A discussion with examples of how the greedy algorithm can fail.)
- https://ocw.mit.edu/courses/sloan-school-of-management/15-053-optimization-methods-in-management-science-spring-2013/lecture-notes/MIT15_053S13_lec17.pdf a slide show from MIT courseware which seems very clear and systematic. It also discusses facility location problems.
- https://diego.codes/post/som-tsp/ Diego Vicente's "Using Self-Organizing Maps to solve the Traveling Salesman Problem" article.

25.4 Where do you go from search

- Machine learning: https://www.youtube.com/watch?v=nKW8Ndu7Mjw
- minimizing "wrongness"
- look at the cookie goodness https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer

CHAPTER

TWENTYSIX

BASIC AGENT-BASED MODELING

Section author: Almond Heil <almondheil@gmail.com>

26.1 Motivation, Prerequisites, and Plan

In this chapter, we will learn the basics of agent-based modeling by creating and customizing a simple model using the Mesa framework. While following along, you should keep in mind that the Mesa framework is one of many ways to approach agent-based modeling in Python.

An agent-based model takes a bottom-up approach to solving a problem, by considering the smallest indiviual members (or agents) that make up a larger system and examining how they interact with each other.

Before you start, make sure you fulfill the following prerequisites.

- The 10-hour “serious programming” course
- Installing the required packages with pip

```
$ pip3 install mesa
```

Now that you’re ready, here’s what we’ll be doing in today’s course!

1. Learn about the basics of agent-based modeling and object-oriented programming
2. Create the classes for a simple infection model
3. Place agents in space and move them
4. Create a visualization of the model
5. Manage infection spread among agents
6. Gather and plot data

26.2 Conceptualizing the model

When building an agent-based model, it’s important to consider the basic building blocks that will make up our model. Based on the broad idea of how diseases spread, we will narrow our focus to how diseases spread by direct contact.

To understand our model as it develops, it’s important to understand a few terms, both from agent-based modeling and object-oriented programming. Some short definitions appear below.

26.2.1 Agent-Based Modeling Concepts

model

An abstraction of reality seeking to distill the behaviors of a complex system so we can understand it more easily.

In Mesa, the model is specifically the structure that manages setting up and running your program, including the agents inside of it.

agent

One of the entities within an agent-based model (wow, what a circular definition!) It can interact with other agents and the world in various ways.

step

A single unit of time in the model. Can also refer to a method an agent follows every time unit.

26.2.2 Object-Oriented Programming Concepts

class

An object-oriented programming concept which refers to a structure containing the framework for making objects–what they can do, what data they hold, etc.

object

One instance of a class, which inherets the structure that's been set up for it. We'll take advantage of this to create many agent objects based on a single class.

method

A function that belongs to a class, and can be called by any object based on that class. For instance, we might expect each agent to be able to move around with an `agent.move()` method.

26.3 Classes and steps

To implement these concepts, we'll create a model. Create a file called `direct_contact.py` and enter this code.

Listing 26.3.1: direct_contact.py

```python
#!/usr/bin/python3

from mesa import Agent, Model
from mesa.time import RandomActivation

class InfectionAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.infected = False

    def step(self):
        print(f"agent {self.unique_id}; infected {self.infected}")


class InfectionModel(Model):
    def __init__(self, N):
        self.num_agents = N
        self.schedule = RandomActivation(self)
```

(continues on next page)

(continued from previous page)

```
        for i in range(self.num_agents):
            a = InfectionAgent(i, self)
            self.schedule.add(a)
    def step(self):
        self.schedule.step()
```

Above, we define two classes. InfectionAgent is based on Mesa's agent class, and we define how it acts when it initializes and when it steps. By using `super().__init__(unique_id, model)` in the InfectionAgent's intialization code, we tell it to take arguments for those two variables from whatever created it. In this case, that means the model that the user defines. We also set the agent's infected status to false, which we'll edit down the line to start an infection.

InfectionModel is based on Mesa's model class. When it initializes, it creates a schedule to run the model with and adds agents to it, passing them the `unique_id` and `model` parameters.

Now, it's time to see the code in action! In your terminal, open a live Python session by typing `python3` and enter the following.

```
>>> from direct_contact import *
>>> model = InfectionModel(10)
>>> model.step()
```

You will see the program output something like this.

```
agent: 1 infection: False
agent: 5 infection: False
agent: 9 infection: False
agent: 8 infection: False
agent: 4 infection: False
agent: 3 infection: False
agent: 7 infection: False
agent: 2 infection: False
agent: 6 infection: False
agent: 0 infection: False
```

If you repeat `model.step()`, you will notice that the order the agents call out is different each time. This is because of the RandomActivation we added which tells the model how to progress when it takes a step.

Exercise 26.1: Scheduling methods To see the difference the scheduling method makes, switch out the random activation we added in favor of `BaseScheduler` or another activation method from the Mesa time module. Notice how this affects the order in which agents act when `model.step()` is called. If you do this, you will also need to import BaseScheduler instead of RandomActivation at the top of the program.

When you want to test changes you've made to your model, make sure to exit your python interpreter with Control+D or by typing `exit()`. Then, start a new session and import the code again to see your most recent changes take effect.

26.4 Space and movement

Now that each agent is able to take steps, we will add space and movement to our model. For this example we will be using a grid for simplicity, as well as Mesa's built-in support for grid visualization.

First, we import the necessary components to our project. In this case we want to use a MultiGrid, so that multiple agents can be on top of each other in the same grid cell.

Listing 26.4.1: direct_contact.py

```python
from mesa.space import MultiGrid
```

Then, we can edit out model's __init__ method to let it take width and height parameters. We also change our method of placing agents to give them random positions using the model's random number generator. This generator functions just like the normal Python random module, but it allows us to easily set seeds if we want to reproduce our results down the line.

Listing 26.4.2: direct_contact.py > InfectionModel

```python
class InfectionModel(Model):
    def __init__(self, N, width, height):
        self.num_agents = N
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, torus=True)

        for i in range(self.num_agents):
            a = InfectionAgent(i, self)
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (x, y))
            self.schedule.add(a)
```

The third argument of MultiGrid represents whether the space is toroidal, meaning that agents who walk of one edge of the grid will reappear on the other side. This emulates an infinite space, and helps us avoid issues with agents hiding in corners or being trapped and unable to move.

To add movement, we need to change what happens when an agent takes a step. First, add the move method, which tells the agent to move to a random cell near itself:

Listing 26.4.3: direct_conract.py > InfectionAgent

```python
def move(self):
    x, y = self.pos
    x_offset = self.random.randint(-1, 1)
    y_offset = self.random.randint(-1, 1)
    new_position = (x + x_offset, y + y_offset)
    self.model.grid.move_agent(self, new_position)
```

Now we've defined how the agent moves, but we don't tell it to do that when `step()` gets called. Go ahead and add the instruction to move, and also update the print statement to tell us the agent's position–right now nothing's going to show up onscreen.

Listing 26.4.4: direct_contact.py > InfectionAgent

```python
def step(self):
    self.move()
    print(f"agent {self.unique_id}; pos {self.pos}; infected {self.infected}")
```

Now, try running the code again–with one difference. Now that the model takes parameters for its width and height, we need to provide those when we create it like so.

```python
>>> from direct_contact import *
>>> model = InfectionModel(10, 30, 20)
>>> model.step()
```

If you're having any issues, go ahead and check your work so far against this example. You can do so using the `diff` tool in the command line. Of course, I'm not going to stop you from copy-pasting this working example, but c'mon.

Listing 26.4.5: direct_contact.py

```python
#!/usr/bin/python3

from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.space import MultiGrid

class InfectionAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.infected = False

    def move(self):
        x, y = self.pos
        x_offset = self.random.randint(-1, 1)
        y_offset = self.random.randint(-1, 1)
        new_position = (x + x_offset, y + y_offset)
        self.model.grid.move_agent(self, new_position)

    def step(self):
        self.move()
        print(f"agent {self.unique_id}; pos {self.pos}; infected {self.infected}")

class InfectionModel(Model):
    def __init__(self, N, width, height):
        self.num_agents = N
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, torus=True)

        for i in range(self.num_agents):
            a = InfectionAgent(i, self)
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (x, y))
            self.schedule.add(a)
```

(continues on next page)

(continued from previous page)

```
    def step(self):
        self.schedule.step()
```

26.5 Visualization

Now we are able to move the agents, but we have no idea of where they are going. Of course, you could add the print statement back in, but with the constant movement and random schedule order it becomes difficult to keep track of what's going on. To make this easier, we want to add visualization.

First, we need to add one instruction to our main `direct_contact.py` file. In the init method for InfectionModel, add the line `self.running = True`. It should now match the code below.

Listing 26.5.1: direct_contact.py > InfectionModel

```
def __init__(self, N, width, height):
     self.num_agents = N
     self.schedule = RandomActivation(self)
     self.grid = MultiGrid(width, height, torus=True)
     self.running = True
```

Now, we need to create a visualizer which will display our model and let us interact with it. In this case, we'll be using Mesa's built-in visualization tools because they're accessible and work well for our purposes. Create a new file called `visualization.py` and add the following to it.

Listing 26.5.2: visualization.py

```
#!/usr/bin/python3

from mesa.visualization.modules import CanvasGrid
from mesa.visualization.ModularVisualization import ModularServer

# change this to match your file name if it's not direct_contact.py!
from direct_contact import *

# The parameters we run the model with.
# Feel free to change these!
params = {"N": 30,
          "width": 50,
          "height": 40}

def agent_portrayal(agent):
    portrayal = {"Shape": "circle",
                 "Color": "grey",
                 "Filled": "true",
                 "Layer": 0,
                 "r": 0.75}
    if agent.infected:
        portrayal["Color"] = "LimeGreen"
        portrayal["Layer"] = 1
    return portrayal
```

(continues on next page)

(continued from previous page)

```
grid = CanvasGrid(agent_portrayal,
                  params["width"],
                  params["height"],
                  20 * params["width"],
                  20 * params["height"])

server = ModularServer(InfectionModel,
                       [grid],
                       "Infection Model",
                       params)
server.launch()
```

There's a lot to unpack in this block of code, because a lot is going on under the hood with Mesa's modules. First, we create a dictionary called `params`. It holds the names and values for each parameter the model takes in its `__init__`. Under the hood, Mesa is unpacking this dictionary to use the values as keyword arguments or kwargs, which are used in `"name": value` pairs to initialize the model.

Next, we define the function `agent_portrayal`. It takes an individual agent from the model as input, and outputs the necessary information to draw the agents. Mesa takes care of its visualization with a web browser window, which handles graphics and user interaction with JavaScript.

Luckily, we don't have to deal with the JavaScript side of the equation because Mesa's CanvasGrid module takes care of it—all you will notice is a new tab in your browser pop up. We only need to pass it the portrayal method to use, the dimensions of the grid, and the pixel size of the grid to be displayed.

Finally, we define the server. It unites several data structures we've already created. The first term is the model to use. The second holds a list of the display methods to use (such as the grid we just defined). The third is simply the title to display, and the fourth is the parameters to run the model with.

See also:

If you're interested in how this system works or want to write your own module you can learn more about it in the Mesa documentation.

The python module CanvasGridVisualization.py feeds our data into the JavaScript module CanvasModule.js, which draws everything in the web server.

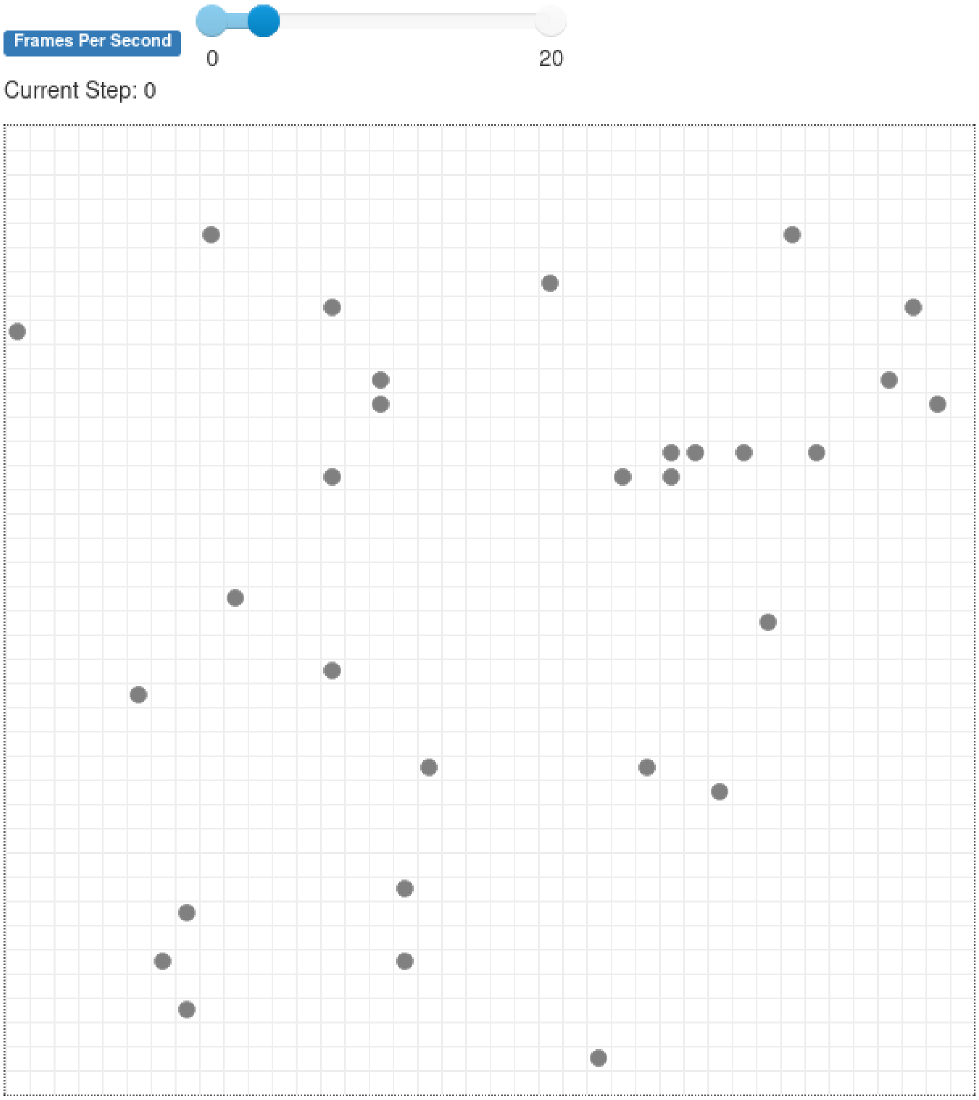
Finally, the python module ModularVisualization.py creates a webserver and passes the relevant config data to your model through it.

With all this done, we can run the model with a single command from the terminal!

```
$ python3 visualization.py
```

Once the server has started it will open a browser window and you can click the "Start" button in the top right to run your model, or the "Step" button move forward incrementaly.

With the server running, you will see a display like this. At this point, you'll only see the agents wandering around, but we'll have them spread infection to each other in the next step.



Current Step: 0

Note: The server won't automatically quit when you click "Stop" or close the browser tab that is displaying it. To stop the model fully, you have to go to the terminal running the model and press Control+C.

26.6 Interactions between agents

First, let's add a method for the InfectionAgent class that allows agents to infect each other. In this method, we use Mesa's built-in `get_neighbors` method to collect a list of all the agents next to a given point. The parameters "moore" and "include_center" specify what counts as a neighboring space. Moore means that diagonal spaces are included, and include_center counts the space that an agent is on.

Listing 26.6.1: direct_contact.py > InfectionAgent

```python
def infect_neighbors(self):
    neighbors = self.model.grid.get_neighbors(self.pos,
```

(continues on next page)

(continued from previous page)

```
                                              moore=True,
                                              include_center=True)
    for neighbor in neighbors:
        if self.random.random() < 0.25:
            neighbor.infected = True
```

Next, we'll edit the agent step method to infect any neighbors only if it is infected.

Listing 26.6.2: direct_contact.py > InfectionAgent

```
def step(self):
    self.move()
    if self.infected:
        self.infect_neighbors()
    print(f"agent {self.unique_id}; pos {self.pos}; infected {self.infected}")
```

Exercise 26.1: Routes of infection As you'll notice, we take an extremely simple aproach to infection: For each of our direct neighbors, we have a 25% chance of infecting them.

What other ways might this disease spread (for instance, only by touch when we stood on the same grid cell as another person)? How might you change the code to reflect these differences?

Finally, we need to add a way for agents to start off infected. Right now we set all agents to be uninfected when initializing the model no matter what, but this means an infection can never start in the model.

First, we'll change the agent's initialization method. Originally we automatically set the agent's infection to False, but now we will take input on whether or not the agent is infected.

Listing 26.6.3: direct_contact.py > InfectionAgent

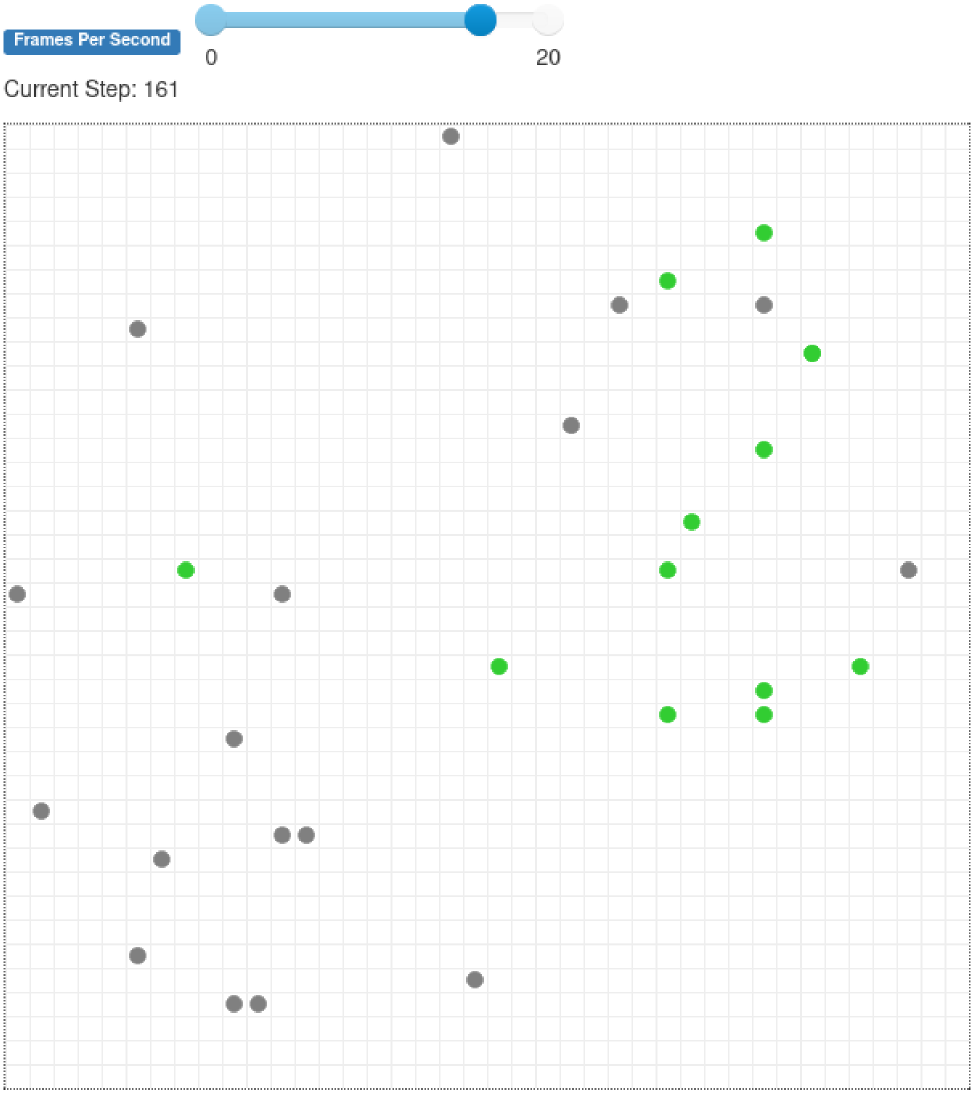
```
def __init__(self, unique_id, model, infected):
    super().__init__(unique_id, model)
    self.infected = infected
```

We'll decide whether the agent is infected when creating it in the model. To do this, we just make the first agent infected by default.

Listing 26.6.4: direct_contact.py > InfectionModel > __init__

```
for i in range(self.num_agents):
    infected = True if (i == 0) else False
    a = InfectionAgent(i, self, infected)
    x = self.random.randrange(self.grid.width)
    y = self.random.randrange(self.grid.height)
    self.grid.place_agent(a, (x, y))
    self.schedule.add(a)
```

When you run the model now, it will look something like this. This screenshot was taken after 161 steps, and the infection has spread to about half of the population.



Current Step: 161

If something isn't running properly, make sure your code matches what's below by running `diff` or using another method.

Listing 26.6.5: direct_contact.py

```python
#!/usr/bin/python3

from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.space import MultiGrid

class InfectionAgent(Agent):
    def __init__(self, unique_id, model, infected):
        super().__init__(unique_id, model)
        self.infected = infected

    def move(self):
        x, y = self.pos
        x_offset = self.random.randint(-1, 1)
        y_offset = self.random.randint(-1, 1)
        new_position = (x + x_offset, y + y_offset)
        self.model.grid.move_agent(self, new_position)

    def infect_neighbors(self):
        neighbors = self.model.grid.get_neighbors(self.pos,
                                                  moore=True,
                                                  include_center=True)
        for neighbor in neighbors:
            if self.random.random() < 0.25:
                neighbor.infected = True

    def step(self):
        self.move()
        if self.infected:
            self.infect_neighbors()

class InfectionModel(Model):
    def __init__(self, N, width, height):
        self.num_agents = N
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, torus=True)
        self.running = True

        for i in range(self.num_agents):
            infected = True if (i == 0) else False
            a = InfectionAgent(i, self, infected)
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (x, y))
            self.schedule.add(a)

    def step(self):
        self.schedule.step()
```

26.7 Data collection & plotting

Now, we're going to use Mesa's built in DataCollector module, which can automatically collect the data for us as we run our model. Of course, we could also collect the data ourselves just by saving it as we run the model, but here we'll use tools from Mesa instead.

26.7.1 Collecting data from the code

To start off, import the data collector into `direct_contact.py`.

Listing 26.7.1: direct_contact.py

```python
from mesa.datacollection import DataCollector
```

Then, we'll create a function **separate** from the agent and model classes which will let us collect the number of infected agents in a model. Next, within the model class, we will initialize a DataCollector and point it to the function we just defined so it can collect data.

Listing 26.7.2: direct_contact.py

```python
def compute_infected(model):
    infected = 0
    for agent in model.schedule.agents:
        if agent.infected:
            infected += 1
    return infected


class InfectionModel(Model):
    def __init__(self, N, width, height):
        self.num_agents = N
        self.schedule = RandomActivation(self)
        self.grid = MultiGrid(width, height, True)
        self.running = True
        self.datacollector = DataCollector(
            model_reporters = {"Infected": compute_infected})

    for i in range(self.num_agents):
        infected = True if (i == 0) else False
        a = InfectionAgent(i, self, infected)
        x = self.random.randrange(self.grid.width)
        y = self.random.randrange(self.grid.height)
        self.grid.place_agent(a, (x, y))
        self.schedule.add(a)

    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)
```

The data collector can also collect two other types of variables, agent-level variables and tables. Model-level variables are like the total number of infected agent's we're now collecting–summaries across the whole model. On the other hand, agent-level variables are unique to each agent. Tables are a bit of a catch-all, and they let you track things that don't match either of those two categories.

26.7.2 Plotting from the command line

Now, we'll collect data from the model–first with a command-line approach. To start off, once again type `python3` in the terminal an import the model as shown. Then, use a loop to step the model up to a certain point.

```
>>> from direct_contact import *
>>> model = InfectionModel(30, 40, 50)
>>> for i in range(400):
...     model.step()
```

Note: In this course, we use a small-scale solution that becomes cumbersome if you try to scale it up. If you want to run several instances and collect data from all of them, you can write your own code to handle the problem or use Mesa's BatchRunner module.

As per usual, your choice! The BatchRunner does a lot of the work for you, but it's worth it to understand what's going on too.

Once the loop has finished running, we want to collect data from the model. By running the code below, we can use the datacollector to generate a Pandas dataframe of the collected data.

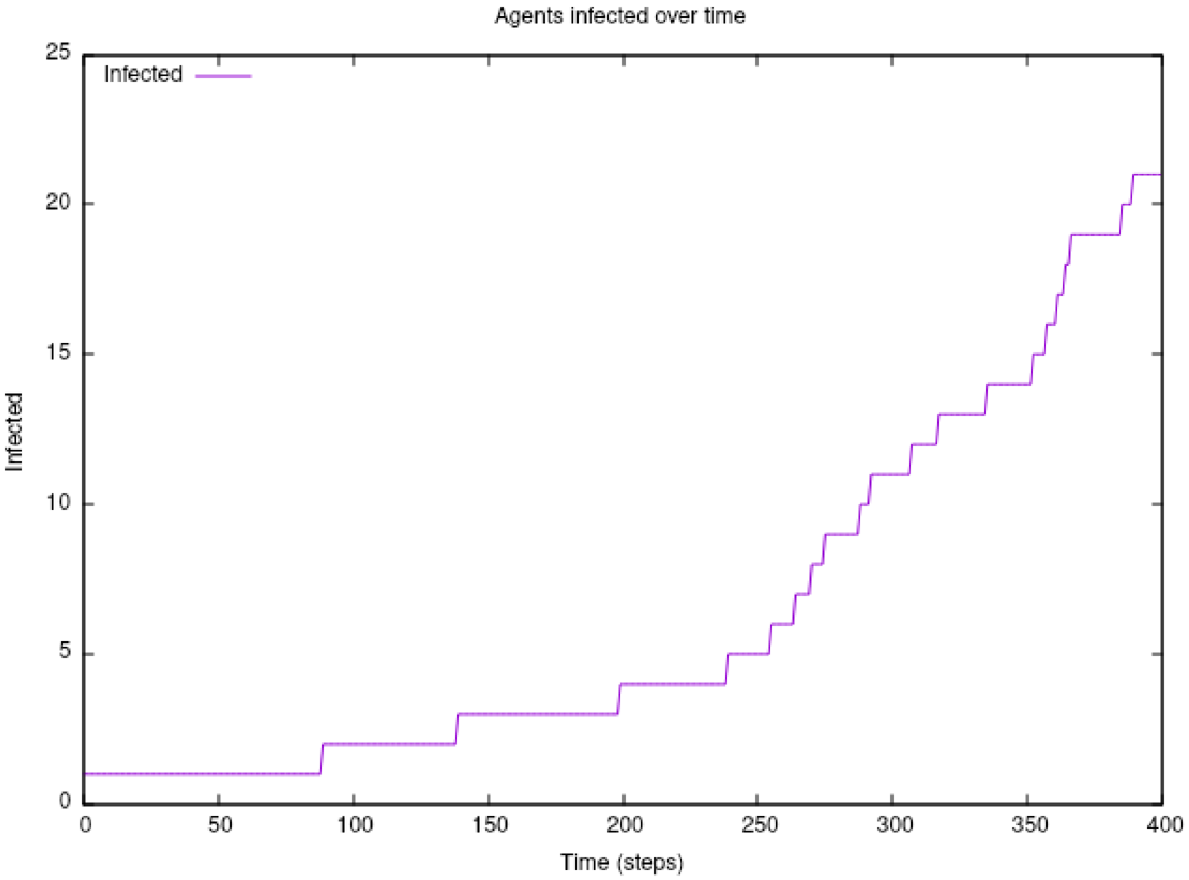
```
>>> data = model.datacollector.get_model_vars_dataframe()
```

Once you have the dataframe, you can do anything you want with it, including plotting it with matplotlib or doing data analysis on the spot. Today, we'll export it to a CSV file and plot it with gnuplot.

```
>>> data.to_csv("model_data.csv", index_label="Steps")
```

Now, exit the python prompt and start a gnuplot session by typing `gnuplot` into your terminal. Enter these lines to generate a graph of the model. It will look something like the graph below.

```
set datafile separator ","
set key autotitle columnhead
set key left top
set xlabel "Time (steps)"
set ylabel "Infected"
set title "Agents infected over time"
plot "model_data.csv" with lines
```



26.7.3 Plotting as the live model runs

If you care less about having workable data and more about getting an idea of the numbers as the model runs, it makes sense to add a graph component to your live visualization. To do this, first import the relevant module in your `visualization.py` file.

Listing 26.7.3: visualization.py

```python
from mesa.visualization.modules import ChartModule
```

Then, create an instance of the ChartModule that will use the data collector to keep track of the agents alive. It's important that the definition of infected_chart goes after the rest of the code but before we define the server, so that it will be loaded when the server starts. Also, make sure to add infected_chart to the list of modules that the server will draw!

Listing 26.7.4: visualization.py

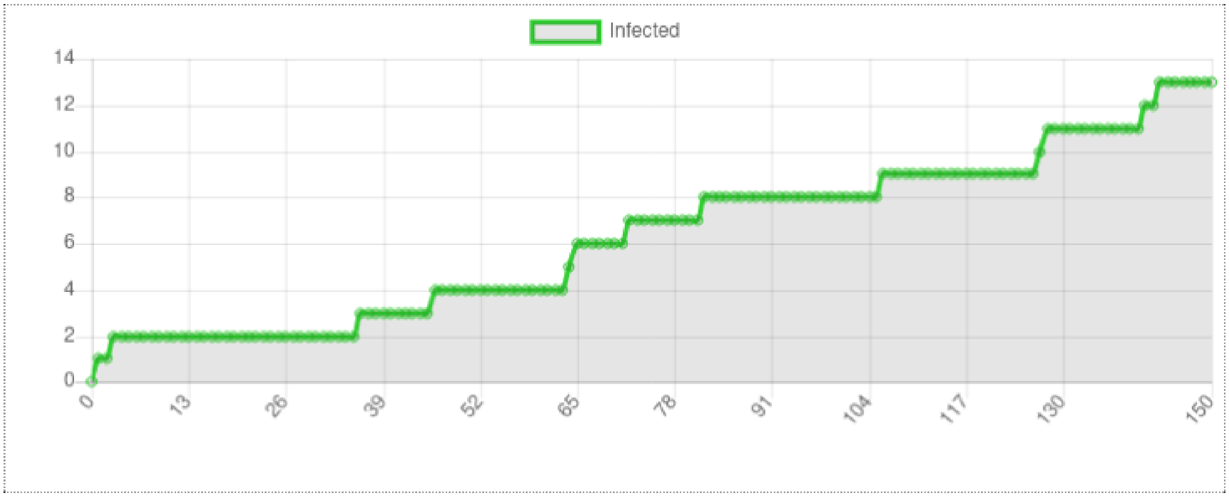
```python
infected_chart = ChartModule([{"Label": "Infected",
                               "Color": "LimeGreen"}],
                             data_collector_name='datacollector')

server = ModularServer(InfectionModel,
                       [grid, infected_chart],
                       "Infection Model",
```

(continues on next page)

(continued from previous page)

```
                    params)
```

Now, when you run the model you will se the usual grid, but there will also be a graph of the number of infected agents. Below is a graph that the model generated after 150 steps.



26.8 Source code

Here are the completed versions of the two files we have used:

`direct_contact.py`

`visualization.py`

Remember to run

```
$ python3 visualization.py
```

to have the code run and visualize in your browser.

26.9 Making an SIR model

Here are files with a partial implementation of an SIR model based on the simple infection model above.

Download these files:

`sir_model.py`

`sir_vis.py`

Remember to run

```
$ python3 sir_vis.py
```

to have the code run and visualize in your browser.

26.10 Further reading

You've completed this course, but there's more to look into in this book and elsewhere if you're interested in agent-based modeling and how agents behave together! Feel free to check out some of the resources below.

- The mini-course in Section **??**, which covers emergent behavior or how complex behavior emerges from simple rules.
- This webpage about SIR and SEIR models for disease. While it uses equations to describe the trends of disease spread, it is possible to extend the model we made here to create a basic SIR model as well!
- The Mesa example boid_flockers, which models bird movement in a flock in continuous space based on Craig Reynolds' boids
- The Wikipedia page and related materials covering Conway's game of life, an excellent example of emergent behavior
- This journal article going in-depth about modeling the dynamics of disease spread. It's particularly interesting to see what they decided to model and what they didn't!

Exercise 26.1: A real SIR model! If you feel up to it, see if you can build on the model we made here to make an actual SIR model, where agents only stay infected for a limited amount of time before being removed from the population (or recovered, if you want to put a positive spin on it.)

There are some ordinary differential equations behind SIR models, so based on your mathematical experience you might feel more or less comfortable dealing with them (I'll admit, they definitely can be scary).

If you're looking for a place to start, try adding a way for agents to become no longer infected by the disease, and see what questions and problems stem off of that.

CHAPTER

TWENTYSEVEN

EMERGENT BEHAVIOR

[status: content-mostly-written]

27.1 Motivation, prerequisites, plan

Purpose: a fun exploration of how complex large-scale behavior emerges from very simple rules on the small scale.

Prerequisites:

- The 10-hour "serious programming" course.

27.2 Before you start

- Watch some of the videos from Melanie Mitchell's course "Introduction to Complexity". At this time only watch the video for Section 1.1 (overall introduction to complexity) and all the videos for Section 6.3 (introduction to 1-dimensional cellular automata): https://www.youtube.com/playlist?list=PLF0b3ThojznRyDQlitfUTzXEXwLNNE-mI
- Read the Nova slides on Everyday Examples of Emergence (4min) http://www.pbs.org/wgbh/nova/nature/emergence-examples.html
- Watch the video "How do schools of fish swim in harmony?" (6min) https://www.youtube.com/watch?v=dkP8NUwB2io

27.3 Write the simple_ca.py program which implements a cellular automaton

You can download the `simple_ca.py` program and we can read through it to see how it works.

27.4 Conway's game of life

You can download the `conway_life.py` program and we can read through it to see how it works.

27.5 Install and run the golly program

Command line: type "golly", it will tell you how to install it. Experiment with drawing your own initial cells and use TAB to advance one at a time, or the run button to advance more.

Then explore the library of starting positions, specifically:

Life -> Guns -> 2c5spaceship-gun-p416

Life -> Guns -> golly-ticker

27.6 Further study

27.6.1 Play with the simple_ca.py program

Make modifications to simple_ca.py to make it

- Run longer runs so you can see more of the patterns.
- Have more cells in each row – make it as wide as your terminal will go.
- You will find the following lines of python code in simple_ca.py:

```
new_row[i] = new_cell(neighbors)
## NOTE: new_cell_with_rule() is part of the extended code (at
## the bottom)
# new_row[i] = new_cell_with_rule(neighbors)
```

  comment out the line that sets new_row[i] to be the result of new_cell(), and uncomment the line that sets it to be new_cell_with_rule(). Then look at the code in new_cell_with_rule() and play with setting different rules and seeing what the cellular automata look like.

27.7 Further reading

- Godel, Escher, Bach

  Read the dialogues "Prelude … Ant Fugue". All the dialogues in the book might be quite interesting to you.

- Watch the rest of the online course on Complexity

  In the prerequisites I had you watch some sections of Melanie Michtell's online course on Complexity. You could watch the rest of the series. At a very minimum watch the sections on genetic algorithms and the game of life.

  https://www.youtube.com/playlist?list=PLF0b3ThojznRyDQlitfUTzXEXwLNNE-mI

CHAPTER TWENTYEIGHT

WEB SCRAPING

[status: mostly-complete-needs-polishing-and-proofreading]

28.1 Motivation, prerequisites, plan

The web is full of information and we often browse it visually with a browser. But when we collect a scientific data set from the web we do not want to have a “human in the loop”, rather we want an automatic program to collect that data so that our results can be reproducible and our procedure can be fast and automatic.

Although my focus here is mainly on scientific applications, web scraping can also be used to mirror a web site.

Prerequisites

- The 10-hour “serious programming” course.
- The “Data files and first plots” mini-course in Section **??**
- You should install the program wget:

  ```
  $ sudo apt install wget
  ```

Plan

Our plan is to find some interesting data sets on the web.

In our first approach in Section **??** we will download them to our disk using the command line program `wget` and plot them with gnuplot. Then in Section **??** we will show how you can retrieve data in your python program.

Finally in Section **??** we will scratch the surface of all the amazing scientific data sets that can be found on the web.

We will try to look at both *time history* and *image* data. Time histories are data sets where we look at an interesting quantity as it changes in time.

Examples of time histories include temperature as a function of time (in fact, all sorts of weather and climate data) and stock market prices as a function of time.

Examples of image data include telescope images of the sky and satellite imagery of the earth and of the sun.

28.2 What does a web page look like underneath? (HTML)

To introduce students to the staples of a web page, remember:

- Not everyone knows what HTML is.
- Few people have seen HTML.

So we introduce HTML (hypertext markup language) by example first, and then point out what "hypertext" and "markup" mean.

So I type up a quick html page, and the students watch on the projector and type their own. The page I put up is a simple hello page at first, then I add a link.

```
<html>
    <head>
        <title>A simple web page</title>
    </head>

    <body>
        <h1>Mark's web page</h1>
        <p>This is Mark's web page</p>
        <p>Now a paragraph with some <i>text in italics</i>
           and some <b>text in boldface</b>
        </p>
    </body>
</html>
```

Save this to a file called, for example, `myinfo.html` in your home directory and then view it by pointing a web browser to `file:///home/MYLOGINNAME/myinfo.html` (yes, there are three slashes in the file URL `file:///...`).

That simple web page lets me explain what I mean by *markup*: bits of text like `<p>` and `<i>` and `<head>` are not text in the document: they specify how the document should be rendered (for example `<b>` and `<i>` specify how the text should look, `<p>` breaks the text into paragraphs). Some of the tags don't affect the text at all, but tell us how the document should be understood (for example the *metadata* tags `<html>` and `<title>`).

Then let's add a hyperlink: a link to the student's school. My html page now looks like:

Listing 28.2.1: A simple web page with an anchor (hyperlink) element in it.

```
<html>
    <head>
        <title>A simple web page</title>
    </head>

    <body>
        <h1>Mark's web page</h1>
        <p>This is Mark's web page</p>
        <p>Now a paragraph with some <i>text in italics</i>
           and some <b>text in boldface</b>
        </p>
        <p>Mark went to high school at
           <a href="http://liceoparini.gov.it/">Liceo Parini</a>
        </p>
```

(continues on next page)

(continued from previous page)

```
    </body>
</html>
```

Then save and reload the page in your browser.

Here I've introduced the *hyperlink*. In HTML this is made up of an element called <a> (anchor) which has an attribute called `href` which has the URL of the hyperlink.

So as we write programs that pick apart a web page we now know what web pages look like. If we want to find the links in a web page we can use the Python string `find()` method to look for `<a` and then for `</a>` and to use the text in between the two.

28.3 Command line scraping with `wget`

In Section **??** we had our first glimpse of the command `wget`, a wonderful program which grabs a page from the web and puts the result into a file on your disk. This type of program is sometimes called a "web crawler" or "offline browser".

wget can even follow links up to a certain depth and reproduce the web hierarchy on a local disk.

In areas with poor network connectivity people can use wget when there is a brief moment of good newtorking: they download all they need in a hurry, then point their browser to the data on their local disk.

28.3.1 First download with wget

Let us make a directory in which to work and start getting data.

```
$ mkdir scraping
$ cd scraping
$ wget https://raw.githubusercontent.com/fivethirtyeight/data/master/alcohol-consumption/
↪drinks.csv
```

We now have a file called `drinks.csv` - how do we explore it?

I would first use simple file tools:

`less drinks.csv`

shows lines like this:

```
country,beer_servings,spirit_servings,wine_servings,total_litres_of_pure_alcohol
Afghanistan,0,0,0,0.0
Albania,89,132,54,4.9
Algeria,25,0,14,0.7
Andorra,245,138,312,12.4
Angola,217,57,45,5.9
## ...
```

If you like to see data in a spreadsheet you could try to use libreoffice or gnumeric:

`libreoffice drinks.csv`

28.3.2 Simple analysis of the drinks.csv file

Sometimes you can learn quite a bit about what's in a file with simple shell tools, without using a plotting program or writing a data analysis program. I will show you a some things you can do with one line shell commands.

Looking at `drinks.csv` we see that the fourth column is the number of wine servings per capita drunk in that country. Let us use the command `sort` to order the file by wine consumption.

A quick look at the `sort` documentation with `man sort` shows us that the `-t` option can be used to use a comma instead of white space to separate fields. We also find out that the `-k` option can be used to specify a key and `-g` to sort numerically (including floating point). Put these together to try running:

```
sort -t , -k 4 -g drinks.csv
```

this will show you all those countries in order of increasing wine consumption, rather than in alphabetical order. To see just the last few 15 lines you can run:

```
sort -t , -k 4 -g drinks.csv | tail -15
```

This is a great opportunity to laugh at the confirmation of some stereotypes and the negation of others.

If you look at the last few lines you see that the French consume the most wine per capita, followed by the Portuguese.

If you sort by the 5th column you will see the overall use of alcohol and the 3rd column will show you the use of spirits (hard liquor) while the 2nd column shows consumption of beer.

28.3.3 Looking at birth data

```
$ wget https://raw.githubusercontent.com/fivethirtyeight/data/master/births/US_births_
↪2000-2014_SSA.csv
$ tr '\r' '\n' < US_births_2000-2014_SSA.csv > births_2000-2014_SSA-newline.csv
$ gnuplot
gnuplot> set datafile separator ","
gnuplot> plot 'births_2000-2014_SSA-newline.csv' using 5 with lines
```

28.4 Scraping from a Python program

28.4.1 Brief interlude on string manipulation

```
$ python3
>>> s = 'now is the time for all good folk to come to the aid of the party'
>>> s.split()
['now', 'is', 'the', 'time', 'for', 'all', 'good', 'folk', 'to', 'come', 'to', 'the',
↪'aid', 'of', 'the', 'party']
# now we've seen what that looks like, save it into a variable
>>> words = s.split()
>>> words
['now', 'is', 'the', 'time', 'for', 'all', 'good', 'folk', 'to', 'come', 'to', 'the',
↪'aid', 'of', 'the', 'party']
>>>
# now try to split where the separator is a comma
>>> csv_str = 'name,age,height'
>>> words = csv_str.split()
```

(continues on next page)

(continued from previous page)

```
>>> csv_str = 'name,age,height'
>>> words = csv_str.split()
>>> words
['name,age,height']
# didn't work; try telling split() to use a comma
>>> words = csv_str.split(',')
>>> words
['name', 'age', 'height']
```

28.4.2 The birth data from Python

Listing 28.4.1: get-birth-data.py - A program which downloads birth data.

```
#! /usr/bin/env python3

import urllib.request

day_map = {1: 'mon', 2: 'tue', 3: 'wed', 4: 'thu', 5: 'fri',
           6: 'sat', 7: 'sun'}

def main():
    f = urllib.request.urlopen('https://raw.githubusercontent.com/fivethirtyeight/data/
↪master/births/US_births_2000-2014_SSA.csv')
    ## this file has carriage returns instead of newlines, so
    ## f.readlines() won't work in all cases.  I read the whole
    ## file in, and then split it into lines
    entire_file = f.read()
    f.close()
    lines = entire_file.split()
    print('lines:', lines[:3])
    dataset = []
    for line in lines[1:]:
        # print('line:', line, str(line))
        line = line.decode('utf-8')
        words = line.split(',')
        # print(words)
        values = [int(w) for w in words]
        dataset.append(values)
    day_of_week_hist = process_dataset(dataset)
    print_histogram(day_of_week_hist)

def process_dataset(dataset):
    ## NOTE: the fields are:
    ## year,month,date_of_month,day_of_week,births
    print('dataset has %d lines' % len(dataset))
    ## now we form a histogram of births according to the day of the
    ## week
    day_of_week_hist = {}
    for i in range(1, 8):
        day_of_week_hist[i] = 0
```

(continues on next page)

(continued from previous page)

```
    for row in dataset:
        day_of_week = row[3]
        month = row[1]
        n_births = row[4]
        day_of_week_hist[day_of_week] += n_births
    return day_of_week_hist

def print_histogram(hist):
    print(hist)
    keys = list(hist.keys())
    keys.sort()
    print('keys:', keys)
    for day in keys:
        print(day, day_map[day], hist[day])

main()
```

28.5 Finding neat scientific data sets

https://www.dataquest.io/blog/free-datasets-for-projects/ (they mention fivethirtyeight)

https://github.com/fivethirtyeight/data

28.5.1 Time histories

Temperature

Births

wget https://raw.githubusercontent.com/fivethirtyeight/data/master/births/US_births_2000-2014_SSA.csv

28.5.2 Images

NASA nebulae

Goes images of the sun

28.6 Beautiful Soup

Beautiful Soup is a powerful python package that allows you to scrape web pages in a *structured* manner. Unlike the code we have seen so far, which does brute-force parsing of html text chunks in Python, beautiful soup is aware of the "document object model" (DOM).

Start by installing the python package. You can probably install with pip, or on debian-based distributions you can run:

```
sudo apt install python3-bs4
```

Now enter the program in Listing **??**:

Listing 28.6.1: Download the Billboard Hot 100 list using Beautiful Soup.

```
#! /usr/bin/env python3

"""This program was inspired by Jaimes Subroto who had written a
program that worked with the 2018 billboard html format.  Billboard
has changed its html format quite completely in 2023, so this is a
re-implementation that handles the new format.
"""

import urllib
from bs4 import BeautifulSoup as soup

def main():
    url = 'https://www.billboard.com/charts/hot-100'
    # url = 'https://web.archive.org/web/20180415100832/https://www.billboard.com/charts/hot-100/'

    # boiler plate stuff to load in an html page from its URL
    url_client = urllib.request.urlopen(url)
    page_html = url_client.read()
    url_client.close()

    # let us save it to a local html file, using utf-8 decoding so
    # that we turn the byte stream into simple ascii text
    open('page_saved.html', 'w').write(page_html.decode('utf-8'))

    # boiler plate use of beautiful soup: use the html parser on the file
    page_soup = soup(page_html, "html.parser")

    # now for the part where you need to know the structure of the
    # html file.  by inspection I found that in 2023 they use <ul>
    # list elements with the attribute "o-chart-restults-list-row", so
    # this is how you find those elements in beautiful soup:
    list_elements = page_soup.select('ul[class*=o-chart-results-list-row]') # *= means contains
    # now that we have our list are ready to read things in, we also prepare
    outfname = 'billboard_hot_100.csv'
    with open(outfname, 'w') as fp:
        headers = 'Song, Artist, Last Week, Peak Position, Weeks on Chart\n'
        fp.write(headers)
        # Loops through each list element
        for element in list_elements:
            handle_single_row(element, fp)
    print(f'\nBillboard hot 100 table saved to {outfname}')

def handle_single_row(element, fp):
    all_list_items = element.find_all('li')
    title_and_artist = all_list_items[4]
    # try to separate out the title and artist.  title should be an
    # <h3> element, artist is a <span> element
    title = title_and_artist.find('h3').text.strip()
```

(continues on next page)

(continued from previous page)

```
    artist = title_and_artist.find('span').text.strip()
    # now the rest of the columns
    last_week = all_list_items[7].text.strip()
    peak_pos = all_list_items[8].text.strip()
    weeks_on_chart = all_list_items[9].text.strip()
    # we have enough to write an entry in the csv file
    csv_line = f'"{title}", "{artist}", {last_week}, {peak_pos}, {weeks_on_chart}'
    print(csv_line)
    fp.write(csv_line + '\n')


if __name__ == '__main__':
    main()
```

If you run:

```
$ chmod +x billboard_hot_100_scraper_2023.py
$ ./billboard_hot_100_scraper_2023.py
```

The results can be seen in the CSV file `billboard_hot_100.csv`:

Table 28.6.1: Billboard Hot 100

Song	Artist	Last Week	Peak Position	Weeks on Chart
Paint The Town Red	Doja Cat	2	1	8
Snooze	SZA	3	2	42
Fast Car	Luke Combs	4	2	27
Cruel Summer	Taylor Swift	6	3	21
I Remember Everything	Zach Bryan Featuring Kacey Musgraves	5	1	5
Last Night	Morgan Wallen	8	1	35
Vampire	Olivia Rodrigo	7	1	13
Fukumean	Gunna	9	4	15
Calm Down	Rema & Selena Gomez	11	3	56
Dance The Night	Dua Lipa	10	6	18
Barbie World	Nicki Minaj & Ice Spice With Aqua	12	7	14
Slime You Out	Drake Featuring SZA	1	1	2
Religiously	Bailey Zimmerman	14	13	21
Sarah's Place	Zach Bryan Featuring Noah Kahan	•	14	1
Flowers	Miley Cyrus	15	1	37
Bad Idea Right?	Olivia Rodrigo	13	7	7
Thinkin' Bout Me	Morgan Wallen	17	9	30
Agora Hills	Doja Cat	•	18	1

continues on next page

Table 28.6.1 – continued from previous page

Song	Artist	Last Week	Peak Position	Weeks on Chart
All My Life	Lil Durk Featuring J. Cole	16	2	20
Need A Favor	Jelly Roll	22	14	26
Anti-Hero	Taylor Swift	26	1	49
Used To Be Young	Miley Cyrus	23	8	5
Rich Men North Of Richmond	Oliver Anthony Music	20	1	7
Greedy	Tate McRae	33	24	2
Kill Bill	SZA	27	1	42
Boys Of Faith	Zach Bryan Featuring Bon Iver	•	26	1
Dial Drunk	Noah Kahan With Post Malone	34	25	15
What Was I Made For?	Billie Eilish	29	14	11
Watermelon Moonshine	Lainey Wilson	35	29	14
Creepin’	Metro Boomin, The Weeknd & 21 Savage	32	3	43
Karma	Taylor Swift Featuring Ice Spice	38	2	29
What It Is (Block Boy)	Doechii Featuring Kodak Black	43	32	21
Great Gatsby	Rod Wave	30	30	2
Get Him Back!	Olivia Rodrigo	21	11	3
I Know ?	Travis Scott	45	11	9
Good Good	Usher, Summer Walker & 21 Savage	57	36	7
Daylight	David Kushner	49	37	24
Peaches & Eggplants	Young Nudy Featuring 21 Savage	42	33	17
Try That In A Small Town	Jason Aldean	47	1	11
Lady Gaga	Peso Pluma, Gabito Ballesteros & Junior H	37	35	14
Qlona	Karol G & Peso Pluma	44	28	7
Meltdown	Travis Scott Featuring Drake	46	3	9
Love You Anyway	Luke Combs	41	15	33
Bongos	Cardi B & Megan Thee Stallion	31	14	3
Deep Satin	Zach Bryan	•	45	1
Boyz Don’t Cry	Rod Wave	25	25	2

continues on next page

Table 28.6.1 – continued from previous page

Song	Artist	Last Week	Peak Position	Weeks on Chart
Save Me	Jelly Roll With Lainey Wilson	58	47	15
Come See Me	Rod Wave	19	19	4
Single Soon	Selena Gomez	54	19	5
Call Your Friends	Rod Wave	18	18	6
Turks & Caicos	Rod Wave Featuring 21 Savage	24	24	2
Hey Driver	Zach Bryan Featuring The War And Treaty	50	14	5
Seven	Jung Kook Featuring Latto	53	1	11
Nine Ball	Zach Bryan	•	54	1
El Jefe	Shakira X Fuerza Regida	•	55	1
All-American Bitch	Olivia Rodrigo	36	13	3
White Horse	Chris Stapleton	68	31	10
Mi Ex Tenia Razon	Karol G	64	22	7
LaLa	Myke Towers	69	43	12
500lbs	Lil Tecca	•	60	1
Tourniquet	Zach Bryan	60	20	5
One More Time	Blink-182	•	62	1
Strangers	Kenya Grace	88	63	2
The Grudge	Olivia Rodrigo	52	16	3
Un Preview	Bad Bunny	•	65	1
Pain, Sweet, Pain	Zach Bryan	•	66	1
Lose Control	Teddy Swims	67	67	7
SkeeYee	Sexyy Red	74	66	4
Everything I Love	Morgan Wallen	77	14	31
Popular	The Weeknd, Playboi Carti & Madonna	72	43	17
HG4	Rod Wave	51	51	2
El Amor de Su Vida	Grupo Frontera & Grupo Firme	92	72	6
Truck Bed	HARDY	82	55	15
Lil Boo Thang	Paul Russell	99	74	2
Long Journey	Rod Wave	39	39	2

continues on next page

Table 28.6.1 – continued from previous page

Song	Artist	Last Week	Peak Position	Weeks on Chart
My Love Mine All Mine	Mitski	•	76	1
Telekinesis	Travis Scott Featuring SZA & Future	78	26	9
Tulum	Peso Pluma & Grupo Frontera	76	43	13
Sabor Fresa	Fuerza Regida	84	26	14
Spotless	Zach Bryan Featuring The Lumineers	70	17	5
Girl In Mine	Parmalee	95	81	9
Deli	Ice Spice	81	41	10
Segun Quien	Maluma & Carin Leon	•	83	1
Lacy	Olivia Rodrigo	59	23	3
Oh U Went	Young Thug Featuring Drake	89	19	14
Nostalgia	Rod Wave & Wet	40	40	2
Johnny Dang	That Mexican OT, Paul Wall & DRODi	91	65	11
HVN On Earth	Lil Tecca & Kodak Black	•	88	1
Bipolar	Peso Pluma x Jasiel Nunez x Junior H	90	60	3
In Your Love	Tyler Childers	85	43	9
Crazy	Rod Wave	48	48	2
Demons	Doja Cat	•	46	2
Making The Bed	Olivia Rodrigo	62	19	3
Logical	Olivia Rodrigo	63	20	3
East Side Of Sorrow	Zach Bryan	75	18	5
Standing Room Only	Tim McGraw	•	61	4
Checkmate	Rod Wave	55	55	2
Can’t Have Mine	Dylan Scott	•	98	1
On My Mama	Victoria Monet	•	98	2
Love Is Embarrassing	Olivia Rodrigo	65	25	3

CHAPTER

TWENTYNINE

GETTING TO PHILOSOPHY

[status: content-mostly-written]

29.1 Motivation, prerequisites, plan

Motivation

Go to any Wikipedia page and follow the first link in the body of its text, and then you follow the first link of that page, and so forth. For almost all Wikipedia pages this procedure will eventually lead you to the Wikipedia page on Philosophy. This observation has its own wikipedia page:

https://en.wikipedia.org/wiki/Wikipedia:Getting_to_Philosophy

Note: When we say "first link" on a wikipedia page, we mean the first link of the article content, after all the "for other uses", "(from Greek . . .)", and other frontmatter – these are not part of the article itself.

This is not a rigorous or deep observation, but it allows us to write some software to analyze and visualize this assertion, and that journey will teach us some very cool programming techniques.

- Explore the "Getting to Philosophy" observation.
- Learn how to do a bit of *web scraping* and text manipulation.
- Use recursive programming for a real world application.
- Learn about the remarkable `graphviz` software.

Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**.
- Recursion from Section **??**.
- Web scraping from Section **??**.

Plan

So how do we write programs that study and visualize this idea? We will:

1. Review what web pages look like.
2. Write programs that retrieve and pick apart web pages looking for links.
3. Learn about graphviz.
4. Use graphviz to analyze the flow of links in our simple web pages.
5. Make those programs more subtle to search through the more complex HTML structure in Wikipedia articles.
6. Output the "first link" chain in various Wikipedia pages to a file so that graphviz can show us an interesting visualization of the that chain.

29.2 Parsing simple web pages

You should quickly review the brief section on what web pages look like in Section **??** before continuing in this section.

Let us start with the simple web page we had in Listing **??** back in Section **??**

Now write a program which finds the first hyperlink in a web page. There are many ways of doing this using sophisticated Python libraries, but we will start with a simple approach that simply uses Python's string methods. An example is in Listing **??**.

Listing 29.2.1: Look through the text of a page for the first hypertext link.

```
#! /usr/bin/env python3

import sys

def main():
    ## get the entire html text from the file
    f = open(sys.argv[1])
    text = f.read()
    f.close()
    pos, link_url = find_first_link(text)
    print('pos, link URL:', pos, link_url)
    print('last_part:', url2last_part(link_url))

    link_ending = url2last_part(link_url)

def find_first_link(text):
    """Finds the first hyperlink in a string of HTML (which might be the
    entire HTML file).  Returns a pair with the position of the first
    link and the href attribute of the link.
    """
    ## search for bits of string that show we're at the start of an anchor
    start_of_anchor = text.find('<a')
    remaining_text = text[start_of_anchor:]
    ## now search for the start of the link, which comes right after href="
    start_of_link = remaining_text.find('href="') + len('href="')
    ## find the end of the link
    text_from_start_of_link = remaining_text[start_of_link:]
```

(continues on next page)

(continued from previous page)

```
    end_of_link = text_from_start_of_link.find('"')
    ## now we have the end of the link, so we take a slice of text
    ## that ends there
    link_url = text_from_start_of_link[:end_of_link]
    ## finally let's keep track of the position in the file of the
    ## start of the link
    link_pos = start_of_anchor + start_of_link
    return link_pos, link_url


def url2last_part(url):
    """Take a URL and pick out the last part, without the .html"""
    url = url.strip()
    last_slash_ind = url.rfind('/')
    last_dot_ind = url.rfind('.')
    if last_slash_ind == -1:
        last_slash_ind = 0
    return url[last_slash_ind:last_dot_ind]

if __name__ == '__main__':
    main()
```

Running this program will give the position and text of the first hyperlink in that HTML file:

```
$ ./find_first_link.py myinfo.html
pos, link URL: 330 myinfo.html
last_part: myinfo
```

29.3 Making vertex and edge graphs

Graph can mean many things. In computer science it is a picture that shows connections between things. The "things" are shown as shapes and the connections are shown as lines or arrows.

There is a very cool program called `graphviz` which lets you make a simple text file and get a graph drawn from it. In Listing **??**: there is a simple example that shows a bit of president Kennedy's family tree:

Listing 29.3.1: The Kennedy family tree

```
## you can process this with
## dot -Tpdf -O kennedys.dot

digraph family_trees {
  Rose_Fitzgerald_Kennedy -> John_Fitzgerald_Kennedy;
  Mary_Josephine_Hannon -> Rose_Fitzgerald_Kennedy;
  Joseph_P_Kennedy_Jr -> John_Fitzgerald_Kennedy;
  Mary_Augusta_Hickey -> Joseph_P_Kennedy_Jr;
  Patrick_Joseph_Kennedy -> Joseph_P_Kennedy_Jr;
  John_F_Fitzgerald -> Rose_Fitzgerald_Kennedy;
}
```

You can then generate the picture with:

```
dot -Tsvg -O kennedys.dot
dot -Tpng -O kennedys.dot
dot -Tpdf -O kennedys.dot
```

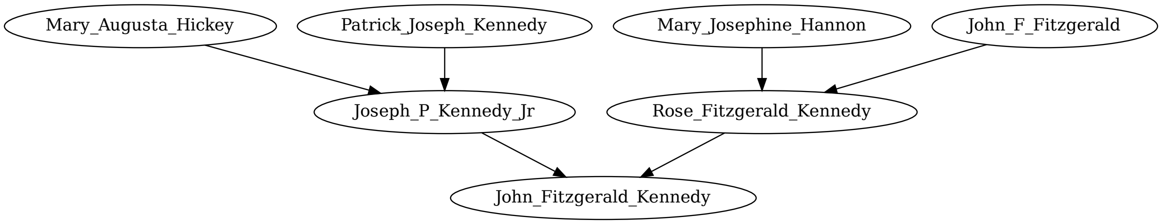


Figure 29.3.1: The immediate family tree of president Kennedy, rendered with graphviz.

You can see more elaborate and sometimes quite visually striking examples at the graphviz web site: http://www.graphviz.org/Gallery.php

You can see that it would be illustrative to make such a graph of the paths through Wikipedia pages.

But first let's take some baby steps: to get more comfortable with how graphviz works, students should create their own `.dot` file with their own family tree. This requires some fast typing, but then they can process it with `dot` and view the picture generated by graphviz.

29.4 A program to get to philosophy

The program I show you here is quite elaborate because it has to deal with some possible scenarios that confuse the issue of which is the "first link" in a wikipedia page. We have provisions that:

- exclude links that come in parentheses
- exclude links before the start of the first paragraph
- exclude links to wikipedia "meta pages", those that start with `File:`, `Help:`, `Wikipedia:` and that end with `.svg`

In Listing **??** we get to see a couple of the types of algorithms we invent as we do this kind of text processing: the code counts the number of open parentheses that have not yet been closed.

Now enter the program in Listing **??**:

Listing 29.4.1: Examine the "Getting To Philosophy" principle on wikipedia.

```
#! /usr/bin/env python3

"""Getting to philosophy: "scrape" a Wikipedia page and follow its
first link recursively to see if you end up at the Philosophy page
https://en.wikipedia.org/wiki/Wikipedia:Getting_to_Philosophy

"""
import urllib.request
import os
```

(continues on next page)

(continued from previous page)

```
import sys

## this is the Philosophy URL -- if we reach this we terminate
# philosophy_list = ['Philosophy', 'Philosophical', 'Existence', 'Semiotics', 'Logic',
↪'Knowledge']
# philosophy_list = ['Philosophy', 'Philosophical', 'Existence',
#                      'Semiotics', 'Semantics', 'Knowledge', 'Linguistics', 'Logic',
#                      'Reasoning']
# philosophy_list = ['Philosophy', 'Philosophical', 'Existence']
philosophy_list = ['Philosophy']

## this is the default list of topics we experiment with
topics_default = ['https://en.wikipedia.org/wiki/Xkcd',
                  'https://en.wikipedia.org/wiki/GNU_Project',
                  'https://en.wikipedia.org/wiki/Bertrand_Russell',
                  'https://en.wikipedia.org/wiki/Plague_of_Justinian',
                  'https://en.wikipedia.org/wiki/Spark_plug',
                  'https://en.wikipedia.org/wiki/Quantum_entanglement',
                  'https://en.wikipedia.org/wiki/Hipparchia_of_Maroneia',
                  'https://en.wikipedia.org/wiki/Toilet_paper']

def main():
    topics = topics_default
    if len(sys.argv) > 1:
        # if user gives URLs on the command line then we use those
        # instead of the default topics
        topics = sys.argv[1:]
    if len(topics) > 1:
        graphviz_fname = 'gtp_graph.dot' # default output file
    else:
        ## if we request a single topic then we can use that as a
        ## filename
        graphviz_fname = topics[0].split('/')[-1] + '.dot'
    print('# GRAPHVIZ_FNAME:', topics, graphviz_fname)
    # canonicalize the filename to remove things like ':' and add .dot
    graphviz_fname = canonicalize_topic(graphviz_fname)
    ## give an error message if the program "dot" (from the package
    ## graphviz) is not available
    if not os.path.exists('/usr/bin/dot'):
        print('Error: the program "dot" does not seem to be installed;')
        print('you can install it with "sudo apt install graphviz"')
        print('and start again')
        sys.exit(1)
    start_graphviz_file(graphviz_fname)
    ## now analyze all the topics
    for topic_url in topics:
        print(f'INITIAL_TOPIC: {url2topic(topic_url)}')
        try:
            url_list = analyze_url([topic_url])
        except RecursionError:
            print(f'Recursion limit exceeded on {topic_url}')
            continue
```

(continues on next page)

(continued from previous page)

```
        except RuntimeError:
            print(f'Recursion limit exceeded on {topic_url}')
            continue
        write_graph_file(url_list, graphviz_fname)
        ## now print some information about what we just did
        print(f'{url2topic(topic_url)} went through {len(url_list)} topics', end="")
        print(f' to reach {(url2topic(url_list[-1]))}')
    ## put the closing line in the graphviz file
    end_graphviz_file(graphviz_fname)
    print('graph information written to file %s' % graphviz_fname)
    ## now run graphviz (the command line is "dot") to make pdf, svg
    ## and png files
    os.system('dot -Tpdf -O %s' % graphviz_fname)
    os.system('dot -Tsvg -O %s' % graphviz_fname)
    os.system('dot -Tpng -O %s' % graphviz_fname)
    print('used "dot" to generate the files %s, %s, %s'
          % (graphviz_fname + '.pdf', graphviz_fname + '.svg',
             graphviz_fname + '.png'))

def analyze_url(urls_so_far):
    """This function analyzes a URL.  We first grab the "next" URL (the
    first link in the page).  If the URL is the arrival point
    (i.e. the Philosophy article) then we return right away with the
    list of URLs visited so far.  If the URL has already appeared
    before then we declare we are in a loop.  If we have had more than
    100 URLs then we return without analyzing further.  The above were
    all terminations, but if *none* of those conditions happen then we
    recursively call this function again to analyze the next URL.
    """
    url = urls_so_far[-1]           # analyze the last one added
    # before we analyze it, first see if they just gave the topic
    # without the full https:// URL
    wikipedia_prefix = 'https://en.wikipedia.org/wiki/'
    if not url.startswith(wikipedia_prefix):
        url = wikipedia_prefix + url
    # then do the analysis recursively
    page_html = urllib.request.urlopen(url).read()
    next_url = analyze_page(url, str(page_html))
    urls_so_far.append(next_url)
    ## print it out - we pad it with zeros and then end it with \r
    ## instead of \n so that we get that cheap animation feel
    # print(f'HOP {len(urls_so_far)} -- {url2topic(next_url)}' + ' '*80,
    #       end="\r")
    # print(f'{url2topic(next_url)} -- HOP {len(urls_so_far)}' + ' '*80,
    #       end="\r")
    print(f'{url2topic(urls_so_far[0])} -- HOP {len(urls_so_far)} -- {url2topic(next_
↪url)}' + ' '*20,
          end="\r")
    if url2topic(next_url).strip('/') in philosophy_list:
        return (urls_so_far)
    elif urls_so_far.count(next_url) > 1:
        return (urls_so_far + urls_so_far[-2:])
```

(continues on next page)

(continued from previous page)

```
    elif len(urls_so_far) > 100:
        return (urls_so_far)
    else:
        return analyze_url(urls_so_far)

def analyze_page(master_url, page_html):
    """Finds the first href (hyptertext link) in the given page."""
    first_href = find_first_href_after_paragraph(master_url, page_html)
    first_href = 'https://en.wikipedia.org%s' % first_href
    return first_href

def find_first_href_after_paragraph(master_url, page_html):
    """Find the first hyperlink after the first <p> tag in the document.
    This is becuase in wikipedia the article content actually starts
    with a <p> tag after all the warnings and other frontmatter have
    been put out.
    """
    # first_p_ind = page_html.find('<p><b>')
    first_p_ind = page_html.find('<p>')
    # print('first_p_ind:', first_p_ind)
    # print(page_html[first_p_ind+3:first_p_ind+6])
    # if page_html[first_p_ind+3:first_p_ind+6] == '<b>':
    #     first_p_ind = page_html.find('<p>', first_p_ind+3)
    html_after_p = page_html[first_p_ind:]
    anchor_split = html_after_p.split('</a>')
    anchor_tag = '<a href="'
    endtag = '"'
    end = 0                   # FIXME: what should the end default be?
    ## FIXME: must exclude the "warning" type of text, which might be
    ## enclosed in this kind of tags: <td class="mbox-text">
    open_parentheses_until_here = 0
    for i, anchor_text in enumerate(anchor_split):
        if anchor_tag in anchor_text:
            # ind = anchor_text.index(anchor_tag)
            base_pos = html_after_p.find(anchor_text)
            pos_after_anchor = anchor_text.find(anchor_tag)
            ## we must also exclude URLs that come up in parentheses,
            ## so we must review all the text leading up to the URL
            ## for open parentheses
            open_parentheses_until_here = count_open_parentheses(master_url, html_after_
↪p,
                                                                 base_pos + pos_after_
↪anchor)
            ## trim the text
            anchor_text = anchor_text[pos_after_anchor + len(anchor_tag):]
            try:
                end = anchor_text.index(endtag)
            except:
                break
        href_url = anchor_text[:end]
        if open_parentheses_until_here > 0:
            continue            # skip anchors that are in parentheses
```

(continues on next page)

(continued from previous page)

```
        ## there only some URLs we consider: those that don't start
        ## with wiki ('cause they point within wikipedia), those that
        ## end with html (otherwise we'd be getting images), ...
        # print('URL_CONSIDERED:', href_url)
        if (href_url.startswith('/wiki/')
            and not href_url.endswith('.svg')
            and not href_url.startswith('/wiki/File:')
            and not href_url.startswith('/wiki/Help:')
            and not href_url.startswith('/wiki/Wikipedia:')):
            # print(f'FOUND_HREF_URL!', href_url, )
            return anchor_text[:end]
    assert(False)               # we should never get here


def write_graph_file(url_list, graphviz_fname):
    """write our list of URLs to a graphviz file"""
    with open(graphviz_fname, 'a') as f:
        prev_topic = url2topic(url_list[0])
        for url in url_list[1:]:
            brief_topic = url2topic(url)
            f.write('    "%s" -> "%s";\n'
                    % (canonicalize_topic(prev_topic),
                       canonicalize_topic(brief_topic)))
            prev_topic = brief_topic
            f.flush()

def start_graphviz_file(fname):
    """put opening information for graphviz at the start of a file"""
    with open(fname, 'w') as f: # zero it out
        f.write('digraph gtp {\n')

def end_graphviz_file(fname):
    """put closing/footer information at the end of a graphviz file"""
    with open(fname, 'a') as f:
        f.write('}\n')

def url2topic(url):
    """Takes a wikipedia URL and strips the boiler plate information to
    give just the name of the topic"""
    # last_slash = url.rfind('/')
    # brief_topic = url[last_slash+1:]
    brief_topic = url.split('/')[-1].strip('/')
    return brief_topic

def canonicalize_topic(topic):
    result = topic
    ## first change the %xx escape sequences used by http URLs back to
    ## their single characters
    result = urllib.parse.unquote(result)
    ## then remove parentheses and hashtags and dashes, replacing them
    ## with underscores
    result = result.replace('(', '_')
```

(continues on next page)

(continued from previous page)

```
    result = result.replace(')', '_')
    result = result.replace('#', '_')
    result = result.replace('-', '_')
    result = result.replace(':', '_')
    return result


def count_open_parentheses(master_url, text, ind):
    """counts how many levels of parentheses are open leading up to this
    index in the text"""
    n_open = 0
    for i, char in enumerate(text[:ind+1]):
        if char == '(':
            n_open += 1
        if char == ')':
            n_open -= 1
    return n_open

main()
```

If you run:

```
$ python3 gtp.py
```

The results can be seen in Figure **??**.

You can also run `python3 gtp.py` with one or more arguments. These arguments can be full Wikipedia URLs or they can be just the final portion. For example:

```
$ chmod 755 gtp.py
$ ./gtp.py https://en.wikipedia.org/wiki/Asterix
$ evince Asterix.pdf &
```

or, alternatively:

```
$ chmod 755 gtp.py
$ ./gtp.py Asterix
$ evince Asterix.pdf &
```

29.5 When things go wrong

Note: Wikipedia pages can change for several reasons. These include the ordinary editing of pages, as well as the *media wiki* software that generates the web site from the original wiki markup.

At this time (2023-10-05) the examples I give below show possible failures in the `gtp.py` program, but at another time these might have been fixed. Still, it is likely that there will always be wikipedia pages that break the assumptions made here.

Ordinary wikipedia articles seem to start the main line text with a <p> element, which has helped us use the simple instruction:

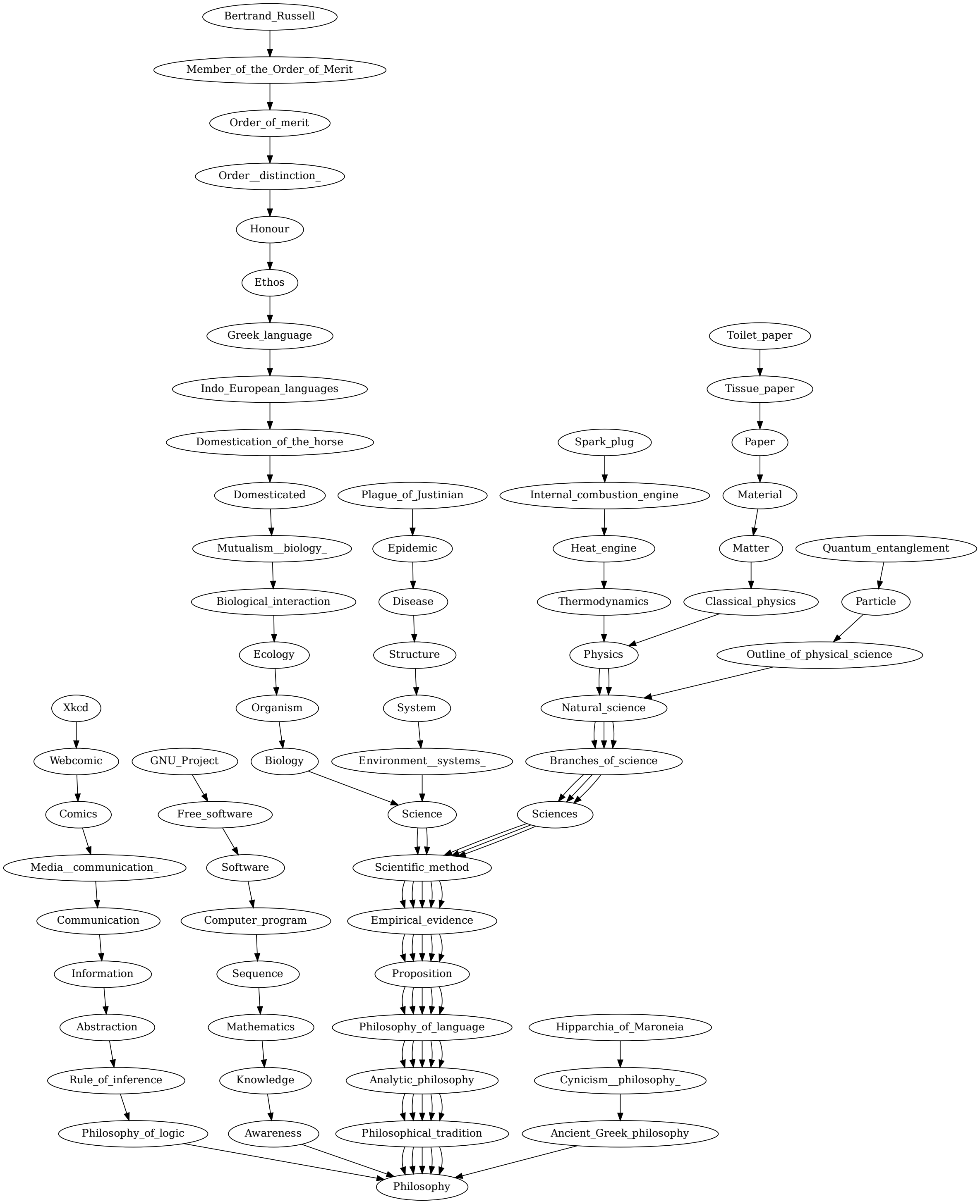


Figure 29.4.1: A graph that shows what happens when you keep clicking the first link in a Wikipedia page. This often ends up in the Wikipedia entry on Philosophy.

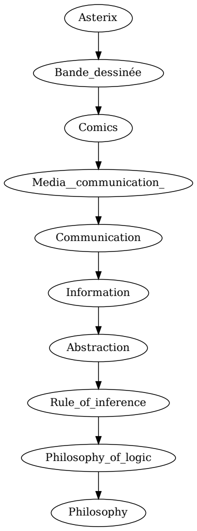


Figure 29.4.2: The chain of first clicks starting at Asterix, obtained with `./gtp.py Asterix` – it is amusing to note that the chain passes through the article on Logic.

```
first_p_ind = page_html.find('<p>')
```

to find the start of the useful text. But some wikipedia pages have a different structure, like list or topic pages.

But even some pages that are not special might break this: at the time of writing this section, the *Complex system* page is organized with a right side bar which has <p> elements in it, and these come before the main text.

So running `./gtp.py Complex_system` goes to `Collective_intelligence` instead of `system` which the ends up taking us into a loop with no progress:

```
$ chmod 755 gtp.py
$ ./gtp.py Complex_system
$ evince Complex_system.pdf &
```

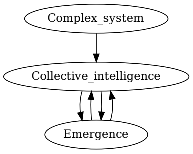


Figure 29.5.1: The chain of first clicks starting at Complex_system, obtained with `./gtp.py Complex_system`. This is a failure of our program:

29.6 When we simply don't "get to philosophy"

Sometimes an article just breaks the mold. At the time in which I wrote an earlier version of this section, Roman_Empire would loop back and forth to Octavian.

While this might be semi-humorously seen as an insightful comment by the "getting to philosophy" meme, it is worth noting that our software had worked well: if you looked at the articles on *Roman Empire* and *Octavian* at that time you would have seen that they do indeed reference each other as first links.

So this was a failure of the meme, not of our program.

As it turns out, at the time of revising (2023-11-06) I find that the Roman Empire article has been revised to start with a link to the Roman Republic, rather than first linking to Octavian. This restores the Getting to Philosophy meme for "Roman Empire", although we can expect this to occur in other articles.

In Figure **??** I show the graph I had gotten at that time.

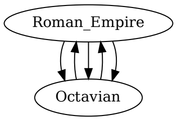


Figure 29.6.1: The chain of first clicks starting at Roman_Empire, obtained with `./gtp.py Roman_Empire` on October 10 2023 when the article had a different first link. This was not a failure of our program: it is simply a different structuring of the articles by their authors.

CHAPTER

THIRTY

MUSIC BASICS

Areas: acoustics, physics, curiosity

[status: just-starting]

30.1 Motivation, prerequisites, plan

As I write in 2020 we listen to recorded music almost exclusively through a computer. It is interesting, instructive, and useful to understand how the computer represents music, how it is stored, compressed, manipulated, and how interesting things get done with it.

To work through this material you should be comfortable with making plots, discussed in Section **??**. You should also install the packages `ffmpeg` and `sox`.

We will start by discussing what sound is. Then we will discuss how it can be represented mathematically. Finally we will look at the various formats that have been devised for computers to store sound files, and how to convert between them.

30.2 What is sound?

Sound is a wave-like sequence of compression and decompression of the air (or other medium). The compression/decompression "front" pushes the next layer of air forward and backwards along the direction of motion. This is called a *longitudinal* wave.

This can be contrasted with different types of waves, like water waves or electromagnetic waves (light, radio, x-rays, ...), where the "up and down" of the wave is perpendicular to the direction in which it moves. Those are called *transverse* waves.

Figure 30.2.1: Longitudinal waves: the expansion/contraction happens along the direction of motion. (Image from wikipedia.)

Figure 30.2.2: Transverse waves: the expansion/contraction happens perpendicular to the direction of motion. (Image from wikipedia.)

Some gentle introductions to sound can be found at:

https://www.mediacollege.com/audio/01/sound-waves.html

https://www.youtube.com/watch?v=qV4lR9EWGlY

In class we discuss the quantities of interest in talking about sound. Some of these are

- amplitude/sound pressure/intensity
- frequency/pitch
- wavelength
- period
- speed
- pure tone versus superposition of frequencies

30.3 How is sound generated?

Ask the class to discuss various ways in which they have seen sound generated.

Some could be: drum head, guitar soundboard, loudspeaker diaphragm, tweeters, wooferes, …

30.4 Measuring and recording

The *human year*, before it has been over-exposed to repetitive sounds, can hear from 20 Hz to 20000 Hz (20 kHz).

Microphones usually try to pick up a very clean (non-distorted) signal in the same frequency.

How do microphones work to translate vibration of air into an electrical voltage that changes in time? The lover's phone, then the carbon microphone, then it gets modern.

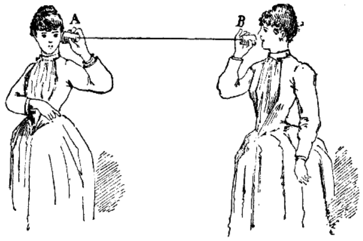


Figure 30.4.1: Robert Hooke's "Lover's Telephone". (Image from wikipedia.)

Figure 30.4.2: A diagram of how the carbon microphon microphone works. When the air compresses it, it conducts more, so you have a higher voltage signal coming out. (Image from wikipedia.)

Can someone research the technical specs of microphones? What is a "frequency response curve"? What would it be for a high quality studio microphone, versus various types of smartphones?

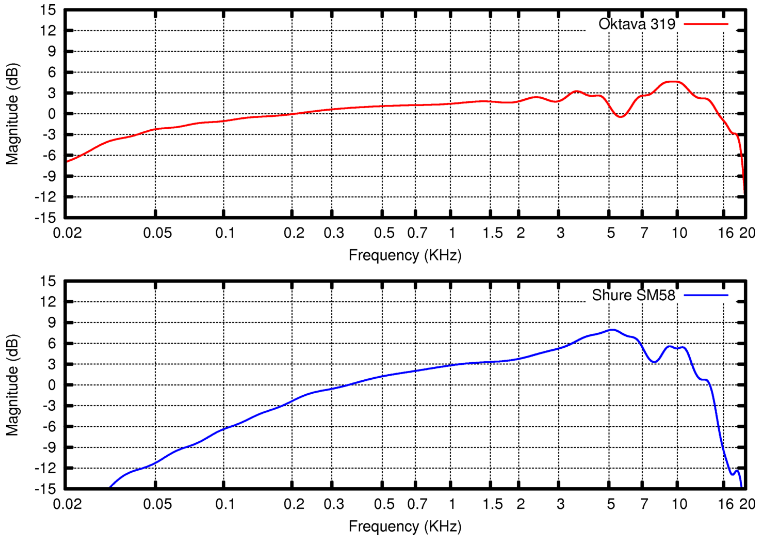


Figure 30.4.3: A “frequency response” curve for two different microphones: the Oktava 319 and the Shure SM58. (Image from wikipedia.)

30.5 What is music

An art form whose medium is sound. Music uses modulations of pitch and amplitude to achieve aesthetical effects.

Discuss some concepts like stereo.

Interesting definitions of “music” proposed by students:

> “A sound that is pleasant, has many different . . . , and doesn’t have to be liked by everyone.”

and

> “Many frequencies that move together in a pattern that makes it pleasant to hear.”

30.6 Understanding what we plot in an *amplitude* plot

Make the following simple plot:

```
$ gnuplot
gnuplot> plot sin(x)
```

That shows a basic $sin()$ wave, but it does not connect to the physical quantities involved. To see how frequency might enter the picture try this out:

```
gnuplot> A = 2.5          # amplitude of 2.5
gnuplot> freq_hz = 440    # 440 hertz - a middle A frequency
gnuplot> set xlabel 'time'
gnuplot> set ylabel 'amplitude'
gnuplot> plot A * sin(2 * pi * freq_hz * x)
```

This frequency is rather high, so the plot not really showing enough information. To see a bit more you can make the gnuplot sampling higher:

```
gnuplot> set samples 10000
gnuplot> plot A * sin(2 * pi * freq_hz * x)
```

Clearly we have to zoom in. To show just a few full periods of the wave let us restrict the domain:

```
gnuplot> plot [-0.01:0.01] A * sin(2 * pi * freq_hz * x)
```

Now we are ready to talk about how to read those axes. Look for the period, understand how the amplitude, frequency, and period appear on it. Discuss why the 2π is in there.

30.7 How does the GNU/Linux microphone work?

We will use the programs `rec` and `play`, both of which are part of the *sox* package in most distributions.. `rec` will record a sound, and `play` will play it back.

As we saw in Section **??**, you can invoke them like this:

```
rec myvoice.dat
```

then speak in to it, or play some music in to it, and hit control-C after just a couple of seconds.

You can play it back with

```
play myvoice.dat
```

If you list your directory you will find that the file *myvoice.dat* has been created, and it has three columns: time, left channel, right channel.

We will plot this file like this:

```
$ gnuplot
gnuplot> plot 'myvoice.dat' using 1:2 with lines   # you can also try 1:3
```

30.8 Generating your own musical tone

30.8.1 A single tone

So how would you generate a tone yourself?

Listing 30.8.1: play_freq.py - play a single note. The one we have put in here is a "middle A (*La*)" which has a frequency of 440 Hz.

```
#! /usr/bin/env python3

"""Demonstrate generating a pure sin() tone, and printing it out in the sox
simple ascii format.

Run this with
./play_freq.py > tone.dat
and play it to the speaker with
play tone.dat
```

(continues on next page)

(continued from previous page)

```
"""

from math import sin, pi


def main():
    play_freq(2, 200.00, 90000, 4) # play 90000 samples at 48kHz
    play_freq(2, 440.00, 90000, 1) # play 90000 samples at 48kHz
    play_freq(2, 523.25, 40000, 3.0) # play 70000 samples at 48kHz
    play_freq(2, 1000.00, 40000, 0.2) # play 70000 samples at 48kHz
    play_freq(2, 261.63, 40000, 0.6) # play 70000 samples at 48kHz
    # you could duplicate this line with further tones, like with
    # frequency 523.25 Hz

    # you could also play a sequence:
    # freq_sequence = [261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25]
    # for freq in freq_sequence:
    #     play_freq(2, freq, 10000)

def play_freq(time, freq, n_samples, amplitude):
    """Plays the note specified by freq, for a duration of n_samples,
    starting at the given time.  Note that if freq is zero then we
    are basically playing a rest note."""
    # 48 kHz seems to be common, and laptop microphones seem to sample
    # at that rate, so let's use it
    sample_rate = 48000
    print('; Sample Rate %d' % sample_rate) # put headers at the top of the file
    print('; Channels 2')

    for i in range(int(n_samples)):
        time = time + 1.0 / 48000.0
        left = amplitude * sin(2*pi*freq*time) # simple sin wave
        right = amplitude * sin(2*pi*freq*time)
        print('%16.10f      %16.10f      %16.10f' % (time, left, right))
    return time

def square_wave(x):
    if x % (2*pi) < pi:
        return 1
    else:
        return -1

main()
```

Put this into a file with:

```
chmod +x play_freq.py
./play_freq > note.dat
play note.dat
```

The frequencies for “do, re, mi, fa, sol, la, si, do” (C,D,E,F,G,A,B,C) are (in Hertz): 261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25.

Note that you could change your main() function to play a full scale of notes, and it might look like this:

Listing 30.8.2: Play a few notes by invoking play_freq() multiple times.

```
def main():
    play_freq(2, 440.00, 70000)    # play 100000 samples at 48kHz
    play_freq(2, 523.25, 70000)    # play 100000 samples at 48kHz
    # freq_sequence = [261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25]
    # for freq in freq_sequence:
    #     play_freq(2, freq, 10000)
```

30.8.2 From notes to frequencies

Let us take the Italian (Do, Re, Mi, Fa, Sol, La, Si) or German/English (A, B, C, D, E, F, G) notation for musical notes and figure out how to convert those into frequencies. This will allow us to write more versatile programs that take a music specification and play it out.

The general mathematical formula is:

$$freq = A4_{freq} * 2^{n_{steps}/12.0}$$

where $A4_{freq}$ is the frequency of the "A above middle C" note, 440 Hz. This is discussed in more detail at https://en.wikipedia.org/wiki/Musical_note#Note_frequency_(hertz)

If we want to convert English

Listing 30.8.3: Convert a note specification (which consists of octave, note, and shartp_or_flat) and generate the frequency of that note.

```
def note2freq(octave, note, sharp_or_flat):
    """Takes a note specification and returns the frequency of that note.
    If note is 'rest' then we return a frequency of zero."""
    ## refer to https://en.wikipedia.org/wiki/Musical_note#Note_frequency_(hertz)
    if note == 'rest':
        freq = 0
    else:
        A4_freq = 440               # A above middle C
        n_steps = note2steps(octave, note, sharp_or_flat)
        freq = A4_freq * math.pow(2, n_steps/12.0)
    return freq
```

This function relies on another function note2steps() which is too long to put here, so we will make a link to a full music generating program generate_music.py which you can study and modify.

You can generate_music.py and save it to a file and play it to your speaker with:

```
chmod +x generate_music.py
./generate_music.py > popcorn.dat
play popcorn.dat
```

30.9 File formats

The *.dat* files we have seen are in the simplest possible format. They are not very expressive and they would become *huge* if we had a long signal. Even those 2-second files were much too big.

We will explore .dat, .au, .aiff, .mp3, .ogg, .webm, .wav, .flac, discussing how each one comes up.

https://en.wikipedia.org/wiki/Timeline_of_audio_formats

https://en.wikipedia.org/wiki/Data_compression#Audio

Section **??**

30.10 Converting our ascii music `.dat` files to other formats

Some of the file formats are very well defined: they can be decoded and played by a program that knows the specification for that format. Sometimes there is even an international expert panel which proposes and maintains the specification for that format. There have been oddities associated with this process: due to an oversight by the mp3 standard group, they allowed the mp3 format to involve a *patented* algorithm, which for a long time made the format unusable by free software. (The patent has expired now.)

The ascii `.dat` files we have been using here are not one of those well-specified formats. As far as we can tell, they are only used by the programs in `sox` (sound exchange) software swite: `rec`, `play`, and `sox`.

On the other hand these ascii files are *extremely* useful for us to understand them, plot them, and write programs that read and write them. Our `play_freq.py` and `generate_music.py` programs generate this format with no effort at all.

To convert our output file `popcorn.dat` (generated in Section **??**) into the more standard `.flac` or `.mp3` formats. The `sox` utility will get us out of the non-standard `.dat` format by turning it into a `.aif` file. From there we can then use the `ffmpeg` program to convert it into dozens of other formats.

For example:

```
./generate_music.py > popcorn.dat
sox popcorn.dat popcorn.aif
ffmpeg -i popcorn.aif popcorn.flac
ffmpeg -i popcorn.aif popcorn.mp3
ls -lsh popcorn.*
```

Here is the output I get from listing those music files in their various formats:

```
5.9M -rw-rw-r-- 1 markgalassi markgalassi 5.9M Jan 14 13:26 popcorn.aif
 45M -rw-rw-r-- 1 markgalassi markgalassi  45M Jan 14 13:26 popcorn.dat
1.1M -rw-rw-r-- 1 markgalassi markgalassi 1.1M Jan 14 13:26 popcorn.flac
252K -rw-rw-r-- 1 markgalassi markgalassi 251K Jan 14 13:27 popcorn.mp3
```

This gives a really interesting look at the effect of using these various file formats. The original `popcorn.dat` file is 45 megabytes in size (this should strike you as way too big). Once you convert to the 1988 vintage *audio interchange file format* (aif) file `popcorn.aif` it is down to about 6 megabytes. The modern *free lossless audio codec* (flac) format is 1.1 megabytes, and if you are willing to lose a small amount of musical quality with the “lossy” *mp3* format you can get it down to a quarter of a megabyte.

You could now play the flac or mp3 file using a music or video program. A quick way from the command line is to run:

```
vlc popcorn.flac
```

30.11 Effects filters

https://linuxgazette.net/issue73/chung.html

CHAPTER

THIRTYONE

COLLECTING MP3S

[status: content-mostly-written]

NOTE: before having anyone work on this project you should make sure that the videos you are downloading from youtube are all "OK" to be downloaded. Check the copyright on the videos and youtube's terms for them.

31.1 Purpose: turn audio from youtube into mp3s

Take some URLs with songs. I will show some that are licensed with an appropriate Creative Commons license, which means you will not get in trouble if you download and reuse this music.

- https://www.youtube.com/watch?v=hz_twrj4fMo
- https://www.youtube.com/watch?v=waoUychgyXs
- https://www.youtube.com/watch?v=-5QvPfj1k_s
- https://www.youtube.com/watch?v=oQpwHMLzwqU
- https://www.youtube.com/watch?v=FHbwpfxARb0

31.2 preparation/prerequisites

- have youtube-dl installed. You can do so with any of the methods described here:

  https://github.com/rg3/youtube-dl/blob/master/README.md#installation

  but since youtube formats move fast, you probably don't want the stock system youtube-dl.

  Easiest is probably to use pip. We do:

```
$ sudo apt-get install python3-pip
$ pip3 install --user --upgrade youtube-dl
```

- the programs easytag and id3info

```
$ sudo apt-get install easytag libid3-tools
```

- ffmpeg for converting formats

```
$ sudo apt-get install ffmpeg
```

- the media player vlc

```
$ sudo apt-get install vlc
```

- a music player. Your system probably come with rhythmbox; another is “clementine”. You can install them with:

```
$ sudo apt-get install rhythmbox

or
```

```
$ sudo apt-get install clementine
```

31.3 Get the video

Grab the video with:

```
$ youtube-dl -t "https://www.youtube.com/watch?v=FHbwpfxARb0"
```

you can see any files that might have been downloaded with:

```
$ ls -sh
```

then canonicalize and clean up the file name a bit with something like:

```
$ mv Hartmut\ Lindemann\ Plays\ Paganini\ -\ Sonata\ per\ la\ Gran\ Viola-FHbwpfxARb0.
↪mp4 Hartmut_Lindemann_Plays_Paganini-Sonata_per_la_Gran_Viola.mp4
```

(Note that that “mv” command could use a lot of “TAB” for filename completion so as to nail the backslashes.)

Now that the filename does not have spaces it is much more manageable.

List again with:

```
$ ls -sh
```

31.4 Verify that it’s a good video file

You can now play this on your own computer (even disconnected from the network!):

```
$ vlc Hartmut_Lindemann_Plays_Paganini-Sonata_per_la_Gran_Viola.mp4
```

31.5 Extracting the audio portion

Now extract the mp3 audio from the video with:

```
$ ffmpeg -i Hartmut_Lindemann_Plays_Paganini-Sonata_per_la_Gran_Viola.mp4 Hartmut_
↪Lindemann_Plays_Paganini-Sonata_per_la_Gran_Viola.mp3
```

And now a listing with

```
$ ls -sh
```

shows that you have the mp3 file.

31.6 Tagging the mp3 file

There is a format called “id3” which lets you put information about a piece into a music file. This allows music players to form a database of your music and to display information about when you play it.

Edit the file with

```
$ easytag Stevie-Wonder_Blowin-in-the-wind-Live.mp3
```

Set the track title and artist.

You can verify that the id3 tags are set with:

```
$ id3info Stevie-Wonder_Blowin-in-the-wind-Live.mp3
```

Now you can put it in your ~/Music/ directory and your music player will pick it up.

Note that you can use the –extract-audio option for youtube-dl to extract the audio immediately, but it might use an obscure format like “.opus”, so you’d still need to use ffmpeg to convert it to mp3.

31.7 A shortcut to the mp3

We wanted to see the details of how we can convert files with ffmpeg, but here is a short cut using some of youtube-dl’s options:

```
youtube-dl -t -f bestaudio --extract-audio --audio-format mp3 --audio-quality 0 'https://
↪www.youtube.com/watch?v=FHbwpfxARb0'

# the next one is considered best practice for title extraction:
youtube-dl -o "%(title)s.%(ext)s" -f bestaudio --extract-audio --audio-format mp3 --
↪audio-quality 0 'https://www.youtube.com/watch?v=3SL0oRcD7t0'
```

See this answer on askubuntu.com:

https://askubuntu.com/questions/634584/how-to-download-youtube-videos-as-a-best-quality-audio-mp3-using-youtube-dl

CHAPTER

THIRTYTWO

COMPUTER ART

[status: content-partly-written]

Sadly the fractal flame screensaver ElectricSheep was made proprietary several years ago, so it cannot grace our computer monitors anymore. But there is much good computer art that can be generated using free software tools.

Here are some references to explore to see if it is possible to develop a chapter for this book on computer art. At this time these references are focused on evolutionary art, but other possibilities should be examined. Connections to 1d and 2d (like Conway's life) cellular automata should be explored.

Preqrequisites:

```
sudo apt install -y geeqie imagemagick
sudo apt install -y gimp
```

32.1 Understanding photos and images

32.1.1 Discussion of graphics formats

Show a PDF and a .png or .jpg and try to zoom in on them. Talk about vector formats vs. raster formats.

32.1.2 Photo collection management

Introduce shotwell and digikam. Urge students to take pictures with a camera or phone, but bring a cheap camera to have students take some 80 photos at the start of class, then download them to the computer with a USB cable.

I have also prepared an archive of NASA's "astronomy picture of the day" (APOD) images dating back to 1994 or so, and put htem in an archive file. You can download them like this:

```
# prepare a space under ~/Pictures for NASA APOD photos
mkdir ~/Pictures/apod
cd ~/Pictures/apod
wget https://www.galassi.org/mark/.tmp/for-courses/apod-just-a.tar
tar -xvf apod-just-a.tar

# after this you could also get a bigger set, if you can let it run
# during dinner or overnight:

wget https://www.galassi.org/mark/.tmp/for-courses/apod-big-archive.tar
tar -xvf apod-big-archive.tar
```

Here is another one that I have put up:

```
# prepare a space under ~/Pictures for stock photos
cd ~/Pictures
mkdir stock
cd stock
# get a couple of sets of images
wget https://www.galassi.org/mark/.tmp/for-courses/eric-kim-stock-photos.zip
unzip eric-kim-stock-photos.zip
```

And one from NASA:

```
# prepare a space under ~/Pictures for other NASA photos
cd ~/Pictures
mkdir nasa
cd nasa
wget https://solarsystem.nasa.gov/system/downloadable_items/1304_JPG.zip
unzip -d NASA_1304 1304_JPG.zip
```

This should be enough to get started. But we can get more from the https://creativity103.com/ web site, using a tip on recursive wget from stackoveflow:

https://stackoverflow.com/questions/4602153/how-do-i-use-wget-to-download-all-images-into-a-single-folder-from-a-url

32.1.3 Image manipulation: command line and GUI

Introduce ImageMagick

Work through some of the filters shown in the ImageMagick cookbook at:

https://www.imagemagick.org/Usage/transform/#art

Start with everyone taking their favorite image. They should name it `myimage-original.jpg`

If you don't have a favorite image handy then you can download NASA's "Pillars of Creation":

```
wget https://upload.wikimedia.org/wikipedia/commons/6/68/Pillars_of_creation_2014_HST_
↪WFC3-UVIS_full-res_denoised.jpg
mv Pillars_of_creation_2014_HST_WFC3-UVIS_full-res_denoised.jpg myimage-original.jpg
```

Then give it a workable size so things dont' take too long for the purpose of this class. I recommend it be approximately 1500x700 pixels in size so that it's not too slow to process, but if students want to use a photo with a modern phone camera (which have many more pixels than that) they can do so and then reduce them with something like:

```
convert -resize 1500x myimage-original.jpg myimage.jpg
```

So from here on let's call our picture myimage.jpg, so rename your picture to that filename.

Next we see how the pixel spread works:

```
convert myimage.jpg -spread 5 myimage-spread-005.jpg
convert myimage.jpg -rotate 5 myimage-rotate-005.jpg
```

Now we'll create a sequence of files with different pixel spreads using a quick inline shell script:

```
for i in `seq -w 0 100`
do
    echo $i
    convert myimage.jpg -spread $i myimage-spread${i}.jpg
done
```

Now make a film with:

```
ffmpeg -framerate 3 -i myimage-spread%03d.jpg spread-movie.mp4
```

and view it with

```
totem spread-movie.mp4 &
```

```
for i in `seq -w 0 360`
do
    echo $i
    convert myimage.jpg -rotate $i myimage-rotate${i}.jpg
done

ffmpeg -framerate 20 -i myimage-rotate%03d.jpg rotate-movie-fr20.mp4
```

Introduce the Gimp

Everyone should know how to start and use the GIMP. Here is a collection of 40 tutorials:

https://www.noupe.com/inspiration/tutorials-inspiration/best-of-gimp-40-professional-tutorials-to-level-up-your-skills-79428.html

We pick one of them and work through part of it with the students.

32.2 metapixel and photomosaics

- Collect a bunch of photos in a hurry, possibly with some cool use of wget and an image search. For example, creativity103.com has archives of images licensed under the "creative commons "attribution" license: https://creativecommons.org/licenses/by/3.0/
- I already showed how to beef up your Pictures folder with commands like `wget -r -np -nc https://creativity103.com/`
- Install metapixel on Linux
- the metapixel tutorial I have found has pointless racy images, so I need to find another one.
- install the webcollage screensaver: `sudo apt install xscreensaver-screensaver-webcollage`
- we have beefed up our personal ~/Pictures directory so we can use that to make the photomosaic database

```
mkdir ~/mp
metapixel-prepare -r ~/Pictures ~/mp --width 32 --height 32
```

once the database is ready we can make our metapixel photo with:

```
metapixel --metapixel --library ~/mp --scale=2 myimage.jpg myimage-mp2.jpg
# and a scaled-by-4 version with:
metapixel --metapixel --library ~/mp --scale=4 myimage.jpg myimage-mp4.jpg
```

Now use geeqie with its “control+scrollwheel” zooming feature to zoom in on the individual tiles in the photomosaic.

32.3 ASCII art

What is ASCII? Explain encoding of characters, and show `man ascii`.

What is ASCII art?

Tools for drawing ASCII art.

Fun and amusing tools. Some of these can be found at https://www.binarytides.com/linux-fun-commands/

```
echo an example of figlet | figlet
banner "have a nice day"
cowsay hey dude
cowsay -f dragon "Run for cover, I feel a sneeze coming on."
cowsay -l
cowsay -f ghostbusters Who you Gonna Call
sl
```

The grand old UNIX *fortune* command is still available, an dyou can pipe it to others:

```
fortune -s     # repeat a few times
fortune -s | cowsay
```

`cmatrix` uses the UNIX *curses* library to draw a matrix animation on your terminal.

Converting raster images to ASCII art. `jp2a` on GNU/Linux systems: let us take pictures of famous computer scientists Margaret Hamilton and Dennis Ritchie:

```
wget https://upload.wikimedia.org/wikipedia/commons/6/6f/Margaret_Hamilton_-_restoration.
↪png
wget https://upload.wikimedia.org/wikipedia/commons/2/23/Dennis_Ritchie_2011.jpg


## make your terminal very big and try
jp2a -f Margaret_Hamilton_-_restoration.png
jp2a -f --color Margaret_Hamilton_-_restoration.png
jp2a -f Dennis_Ritchie_2011.jpg
jp2a -f --color Dennis_Ritchie_2011.jpg
```

To see jp2a work better, make your terminal have really tiny fonts (in many terminal programs you can do this with `ctl+-`), then resize the window to be really big, and then run the jp2a commands again. You might be astonished at the results.

emacs has an ASCII art drawing mode. It might be worth exploring.

Joyce Levine has pointed me to tilde town: https://tilde.town/ which might be worth exploring.

There was a famous ascii art movie of Star Wars that one could reach with

```
telnet towel.blinkenlights.nl
```

but it has gone offline recently. You can find videos of it on youtube. A real tour-de-force. According to this thread on reddit it has been taken down:

https://www.reddit.com/r/sysadmin/comments/pbv7xm/is_towelblinkenlightsnl_dead/

but you can get it (without the fancy ascii colors) by typing a telnet command followed by typing starwars:

```
telnet telehack.com
starwars
```

A discussion of the ascii animation used there can be found at:

https://www.asciimation.co.nz/index.php

https://www.asciimation.co.nz/asciimation/ascii_faq.html

A few more things to type:

```
sl
fortune
factor 12103  # factoring numbers? can we use this to search for Mersenne primes?
factor `echo "2^7-1" | bc` ; factor `echo "2^11-1" | bc` ; factor `echo "2^13-1" | bc`
pi 50
espeak "Hello Linux, where are the penguins"
aafire -driver curses
```

32.4 Evolutionary art

- https://en.wikipedia.org/wiki/Electric_Sheep
- https://en.wikipedia.org/wiki/Software_art
- https://en.wikipedia.org/wiki/Evolutionary_art
- https://iasl.uni-muenchen.de/links/GCA-IV.3e.html
- transcend https://transcend.sourceforge.net/

32.5 Image manipulation from your own Python program

We will learn to use Pillow, the python imaging library (PIL).

At the shell:

```
sudo apt install python3-pil
```

Also give yourself a photo to work with. For this tutorial let's keep it small so it's faster and views better. You can use a command like:

```
convert -resize 800 my_big_picture.jpg myimage.jpg
```

In the python interpreter, following the tutorial at https://pillow.readthedocs.io/en/stable/handbook/tutorial.html

```
## import the Image portion of the PIL library
from PIL import Image
## load an image
im = Image.open('myimage.jpg')
## show its information
print(im.format, im.size, im.mode)
## show the picture itself
im.show()
## then close that window
```

We now have an image stored in the object `im` and we can work on manipulating it.

32.5.1 Geometric transformations

```
import os, sys
from PIL import Image

size = 128, 128

infile = 'myimage.jpg'
outfile = os.path.splitext(infile)[0] + '.thumbnail' + '.jpg'
print(outfile)
thumb = Image.open(infile)
thumb.thumbnail(size)
thumb.save(outfile, 'JPEG')
```

```
## copy a rectangle from the image
box = (300, 300, 600, 600)
region = im.crop(box)

## now play with that rectangle, then paste it back in
region = region.transpose(Image.ROTATE_90)
## now paste it back in
im.paste(region, box)
## now box is the image with the small region we chose
im.show()
im.save('myimage_pasted.jpg', 'JPEG')
```

You will now see that the region we cut out of the photo has been rotated 90 degrees.

```
## reload the image from scratch
im = Image.open('myimage.jpg')
## rotate 45 degrees counterclockwise
rotated = im.rotate(45)
rotated.save('myimage_rotated_45.jpg', 'JPEG')
## now make an animation
various_rotations = []
for i in range(360):
    rotated = im.rotate(i)
    fname = 'myimage_rotated_%03d.jpg' % i
    print('writing out %s' % fname)
    rotated.save(fname, 'JPEG')
```

If you type `ls` you will see that you now have many files with names like `myimage_rotated_17.jpg`. You can use `geeqie` to view them and even get a crude animation by holding the space bar down. You can also use what we learned in Section **??** and try this:

```
## make a movie with:
ffmpeg -framerate 24 -i myimage_rotated_%03d.jpg myimage_animated.mp4
## view it with:
vlc myimage_animated.mp4 &
```

32.5.2 Filters and enhancement

```
from PIL import Image
from PIL import ImageFilter
## load an image
im = Image.open('myimage.jpg')
## out = im.filter(ImageFilter.DETAIL)
## out.save('myimage_detail.jpg', 'JPEG')
out = im.filter(ImageFilter.CONTOUR)
out.save('myimage_contour.jpg', 'JPEG')
out = im.filter(ImageFilter.EMBOSS)
out.save('myimage_emboss.jpg', 'JPEG')
out = im.filter(ImageFilter.FIND_EDGES)
out.save('myimage_edges.jpg', 'JPEG')
out = im.filter(ImageFilter.SMOOTH)
out.save('myimage_smooth.jpg', 'JPEG')
out = im.filter(ImageFilter.BLUR)
out.save('myimage_blur.jpg', 'JPEG')

## NOTE: in the next one we start from "out", the smoothed photo
restored = out.filter(ImageFilter.EDGE_ENHANCE)
restored.save('myimage_restored.jpg', 'JPEG')

out = im.filter(ImageFilter.SMOOTH_MORE)
out.save('myimage_smooth_more.jpg', 'JPEG')
```

Now compare all the images we produced. In particular, look carefully at `myimage_smooth.jpg` and `myimage_restored.jp` to see how the EDGE_ENHANCE transformation takes away the fuzziness that had been introduced by SMOOTH.

32.6 Topics for further study

Another Pillow tutorial which starts slowly but gets into interesting examples:

https://www.tutorialspoint.com/python_pillow/python_pillow_quick_guide.htm

https://realpython.com/fingerprinting-images-for-near-duplicate-detection/

https://www.imagemagick.org/script/compare.php

https://askubuntu.com/questions/209517/does-diff-exist-for-images

https://jeremykun.com/2012/01/01/random-psychedelic-art/

Take one of those classic old “over the top” movie transition effects, or a modern parody of one, like this one from Spinal Tap: https://youtu.be/QrJlyapt6OY?t=114

Try to take an image and create your own over the top transition using Pillow.

CHAPTER

THIRTYTHREE

IMAGE FILTERING

[status: in progress]

33.1 Motivation, prerequisites, plan

33.1.1 Motivation

It hardly seems worth mentioning: nowadays image filtering seems to come up everywhere. We have explored some command-line filtering when looking at computer art in Section **??**. Here we will look at manipulating images from Python programs. Some techniques we will learn are:

- Blurring images.
- Sharpening images.
- Finding features in images.

33.1.2 Prerequisites

- The 10-hour "serious programming" course.
- The "Data files and first plots" mini-course in Section **??**
- Installing the python3-numpy, python3-scipy and python3-pil packages.
- Being comfortable with user-level programs which manipulate images. Specifically, you should learn to run `eog`, `geeqie`, `convert` and `gimp`.

33.1.3 Plan

We will first explore the filtering of images that can be done with ImageMagick on the command line. Then on to using the Pillow/PIL libraries to manipulate images in Python. Then we will use OpenCV for similar manipulation of static images.

Finally we will move on to getting comfortable with tensorflow, the machine learning library which can tackle so many areas of AI. We will use it to find objects in images and in video streams.

33.2 Manipulating images with command line programs

FIXME: to be written.

33.3 How computers store images, disk and memory

We have frequently seen files saved in a variety of image formats. Probably the most common are `.png` and `.jpg` files, although there are many other image formats as well.

When images are stored *in memory* the format can be quite different. We are more interested in the speed with which we our program can *process* the image, rather than its long term storage requirement on disk.

We seldom deal directly with the image storage: we almost always use function calls from a library. In our case we will use the Python Imaging Library Pillow (formerly called PIL), which has uses an abstract representation of the image in memory. This means that we do not have to understand the details of how it is stored.

33.4 First example: blurring and other effects with PIL

Here we show how to use the Pillow library (the new version of PIL, the Pythin Imaging Library). First we install the library with:

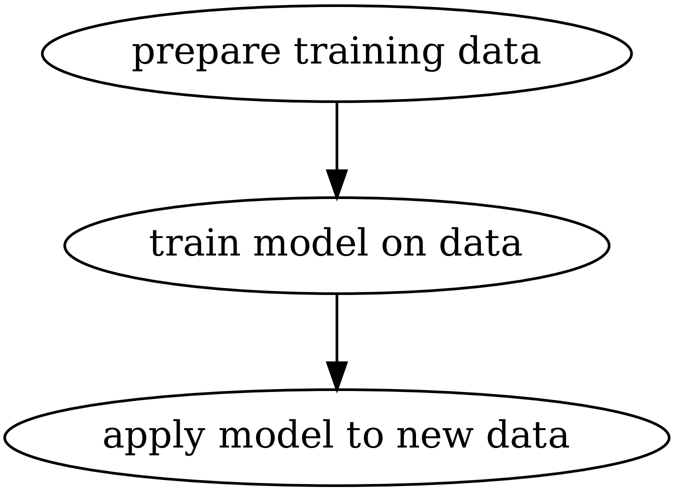
```
pip3 install Pillow
```

Then a simple example using an image file I got off the web. Try to get your own, for example do a web image search for “person with cat” and make sure you search for images that are “Labeled for reuse with modification”. My image is called `person-cat-small.jpg`

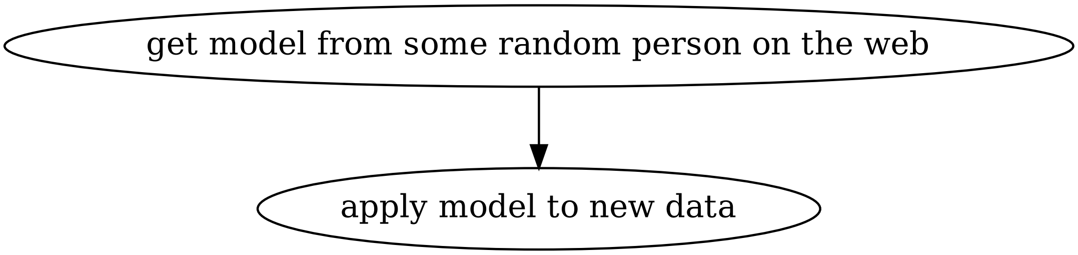
```
from PIL import Image, ImageFilter  # imports the library

im_fname = 'person-cat-small.jpg'
original = Image.open(im_fname) # load an image from the hard drive
blurred = original.filter(ImageFilter.BLUR) # blur the image
embossed = original.filter(ImageFilter.EMBOSS) # emboss the image
contours = original.filter(ImageFilter.CONTOUR) # find contours

for img in (original, blurred, embossed, contours):
    img.show() # display all images
```

33.5 The cycle of training and running an AI system

The general cycle is that you train your model, and then apply it to data:



But sometimes you get a model which someone else has trained, so the cycle is:



33.6 Miscellaneous examples in various areas

33.6.1 Astronomy example with scipy image kit

You can find information on how to use Python to do astronomical image processing with Pillow, scikit-iamge and pyfits at:

http://prancer.physics.louisville.edu/astrowiki/index.php/Image_processing_with_Python_and_SciPy

And this article discusses the skyimage library used with matplotlib, including :

https://www.analyticsvidhya.com/blog/2014/12/image-processing-python-basics/

Both of those articles discuss finding stars within the images.

33.6.2 Extracting the portion of a scan which has text

https://www.danvk.org/2015/01/07/finding-blocks-of-text-in-an-image-using-python-opencv-and-numpy.html

33.6.3 Thresholding

https://pythontic.com/image-processing/pillow/thresholding

33.7 Learning OpenCV

On a GNU/Linux system you can install OpenCV with

```
sudo apt install python3-opencv
```

33.7.1 numpy and opencv

https://medium.com/@manivannan_data/drawing-image-using-numpy-and-opencv-565abdbb3670

33.7.2 Image manipulation with OpenCV

Many tutorials on OpenCV are at:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_watershed/py_watershed.html

Rotation

Let us use examples from:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html#geometric-transformations

Assuming your picture is called `person-cat-small.jpg` you can try:

```
import cv2
import numpy as np

img = cv2.imread('person-cat-small.jpg', 0)

# show original image
cv2.imshow('img', img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# now rotate it
rows,cols = img.shape
M = np.float32([[1,0,100],[0,1,50]])    ## make a rotation matrix
dst = cv2.warpAffine(img,M,(cols,rows))

# then show it
cv2.imshow('img',dst)
```

(continues on next page)

(continued from previous page)

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Finding objects

Some OpenCV sample images can be found at:

https://github.com/opencv/opencv/tree/master/samples/data

We download box.png and box_in_scene.png:

```
wget https://raw.githubusercontent.com/opencv/opencv/master/samples/data/box.png
wget https://raw.githubusercontent.com/opencv/opencv/master/samples/data/box_in_scene.png
```

Now following this page:

https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html#matcher

(but with some adjustments for OpenCV version 3, like ORB() -> ORB_create() and adding a None argument in the drawMatches() routine)

Try this:

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img1 = cv2.imread('box.png',0)          # queryImage
img2 = cv2.imread('box_in_scene.png',0) # trainImage

# Initiate SIFT detector
orb = cv2.ORB_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches[:10], None, flags=2)

cv2.imwrite('object-matches.png', img3)

plt.imshow(img3),plt.show()
```

33.8 Using tensorflow with ImageAI to find objects

ImageAI offers the dream of a brief (they claim 10 lines!) python program that finds objects in images.

To install needed software use:

```
pip3 install tensorflow
pip3 install opencv-python
pip3 install keras
pip3 install imageai --upgrade
```

Let's also mention what some of these components are:

tensorflow

Google's widely used machine learning library.

OpenCV

A library for computer vision which allows the analysis of video streams

Keras

A Python library offering an abstraction of the machine learning

33.8.1 ImageAI + tensorflow from Fritz AI article

Tutorials is at:

https://heartbeat.fritz.ai/detecting-objects-in-videos-and-camera-feeds-using-keras-opencv-and-imageai-c869fe1ebcdb

They have you download a data set with a model and a video:

```
wget https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/yolo.h5
wget https://github.com/OlafenwaMoses/IntelliP/raw/master/traffic-mini.mp4
```

From a fixed video file

Then put this program in a file called `FirstVideoDetection.py`

Listing 33.8.1: Program which analyzes a video stream for objects.

```
#! /usr/bin/env python3
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

# input_video_path = os.path.join( execution_path, "improv.mp4")
input_video_path = os.path.join( execution_path, "traffic-mini.mp4")
# input_video_path = os.path.join( execution_path, "godzilla.mp4")
output_video_path = input_video_path[:-4] + '_detected_1'
```

(continues on next page)

(continued from previous page)

```
video_path = detector.detectObjectsFromVideo(input_file_path=input_video_path,
                                             output_file_path=output_video_path,
                                             frames_per_second=29, log_progress=True)
print(video_path)
```

Now you can run it with:

```
$ chmod +x FirstVideoDetection.py
$ ./FirstVideoDetection.py
```

It will produce a file called `traffic_mini_detected_1.avi` which you can view with your favorite video viewer – for example:

```
vlc traffic_mini_detected_1.avi
```

So that finds objects in moving images!

From your computer's camera

This is much more exciting.

Put this program in a file called `FirstCameraDetection.py`

Listing 33.8.2: Program which analyzes a camera stream for objects.

```
#! /usr/bin/env python3

from imageai.Detection import VideoObjectDetection
import os
import cv2

execution_path = os.getcwd()

camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath(os.path.join(execution_path , "yolo.h5"))
detector.loadModel()

output_video_path=os.path.join(execution_path, "camera_detected_1")
video_path = detector.detectObjectsFromVideo(camera_input=camera,
                                             output_file_path=output_video_path,
                                             frames_per_second=29, log_progress=True)
print(video_path)
```

Now you can run it with:

```
$ chmod +x FirstCameraDetection.py
$ ./FirstCameraDetection.py
```

Note: You will have to interrupt the program yourself when you want to start collecting video. You can do this with control-c or control-

It will produce a file called `camera_detected_1.avi` which you can view with your favorite video viewer – for example:

```
vlc camera_detected_1.avi
```

So that finds objects in moving images!

The web page referenced above describes the step-by-step explanation of what's being done by the ImageAI and Tensorflow libraries as you run both of these programs.

33.8.2 ImageAI + tensorflow from towarddatascience

Tutorial is at:

https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606

They use a pre-defined model that finds people and vehicles and backpacks. This is in the file `resnet50_coco_best_v2.0.1.h5`

We must get the hdf5 file with model weights:

```
wget https://github.com/OlafenwaMoses/ImageAI/releases/download/1.0/resnet50_coco_best_
↪v2.0.1.h5
```

```
from imageai.Detection import ObjectDetection
import os

execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.0.1.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
↪"image.jpg"), output_image_path=os.path.join(execution_path , "imagenew.jpg"))

for eachObject in detections:
    print(eachObject["name"] , " : " , eachObject["percentage_probability"] )
```

33.9 Using tensorflow from their own tutorials

Warning: At this time the versions of all the libraries are not working well at crucial points where you save and reload a model.

33.9.1 The tutorial from tensorflow.org

Install with:

```
pip3 install tensorflow
pip3 install numpy
pip3 install scipy
pip3 install pillow
pip3 install matplotlib
pip3 install h5py
pip3 install keras=2.3.1
```

Following:

https://www.tensorflow.org/tutorials/quickstart/beginner

Now run this in python:

```
import tensorflow as tf

# set eager execution - we will need it when we call model(...).numpy()
tf.enable_eager_execution()

# Load and prepare the MNIST dataset. Convert the samples from
# integers to floating-point numbers:

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Build the tf.keras.Sequential model by stacking layers. Choose
# an optimizer and loss function for training:

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
    ])

# For each example the model returns a vector of "logits" or
# "log-odds" scores, one for each class.

predictions = model(x_train[:1]).numpy()
predictions

# result should be:
# array([[-0.40252417, -0.36244553, -0.6254247 ,  0.3470046 ,  0.53377753,
#         -0.25291196,  0.42313334, -0.85892683,  0.16624598,  0.01534149]],
#       dtype=float32)

# The tf.nn.softmax function converts these logits to
# "probabilities" for each class:

tf.nn.softmax(predictions).numpy()

# result should be:
```

(continues on next page)

(continued from previous page)

```
# array([[0.06724608, 0.06999595, 0.05380994, 0.14229287, 0.17151323,
#         0.07809851, 0.15354846, 0.04260433, 0.11876287, 0.10212774]],
#       dtype=float32)

# Note: It is possible to bake this tf.nn.softmax in as the
# activation function for the last layer of the network. While
# this can make the model output more directly interpretable,
# this approach is discouraged as it's impossible to provide an
# exact and numerically stable loss calculation for all models
# when using a softmax output.

# The losses.SparseCategoricalCrossentropy loss takes a
# vector of logits and a True index and returns a scalar loss
# for each example.

loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

# This loss is equal to the negative log probability of the true
# class: It is zero if the model is sure of the correct class.

# This untrained model gives probabilities close to random (1/10
# for each class), so the initial loss should be close to
# -tf.log(1/10) ~= 2.3.

loss_fn(y_train[:1], predictions).numpy()

# should be 2.5497844

model.compile(optimizer='adam',
              loss=loss_fn,
              metrics=['accuracy'])

# The Model.fit method adjusts the model parameters to minimize the
# loss:

model.fit(x_train, y_train, epochs=5)

# The Model.evaluate method checks the models performance, usually
# on a "Validation-set" or "Test-set".

model.evaluate(x_test,  y_test, verbose=2)

# The image classifier is now trained to ~98% accuracy on this
# dataset. To learn more, read the TensorFlow tutorials.

# If you want your model to return a probability, you can wrap the
# trained model, and attach the softmax to it:

probability_model = tf.keras.Sequential([
                             model,
                             tf.keras.layers.Softmax()
                             ])
```

(continues on next page)

(continued from previous page)

```
probability_model(x_test[:5])
```

33.9.2 For more on training the network

https://www.datacamp.com/community/tutorials/tensorflow-tutorial

33.9.3 The most complete tutorial on preparing training sets and doing the training

Run through this example of cats and dogs:

https://www.tensorflow.org/tutorials/images/classification

It all works. There is only one typo where they have “accuracy” instead of “acc” in a few places, and it’s easy to fix.

It is quite long because it trains poorly and then really well.

Save with:

os.system(“mkdir -p saved_models/”) model.save(‘saved_models/cats_dogs_initial.h5’) model_new.save(‘saved_models/cats_dogs_improved.h5’)

CHAPTER

THIRTYFOUR

CRYPTOGRAPHY

[status: content-mostly-written]

We will discuss a narrow slice of cryptography. We will start with a weak approach, and then look at how you could get a stronger encryption with key exchange and a pseudorandom number generator.

34.1 Preliminary: ASCII values

A key aspect of some of the techniques we will use is that each character correponds to an integer value. In English such integers are called the *ASCII values* of characters, and they are sequential for sequential letters. ASCII stands for "American Standard Code for Information Interchange".

To familiarize yourself with the ASCII codes for each character take a look at the man page for ascii:

```
man ascii
```

and pay attention to the decimal and hex encoding of the various characters. You will recognize digits (ascii values 48 to 57), upper case letters (65 to 90) and lower case letters (97 to 122). There are also many standard keyboard symbols as well as control characters.

In Python you convert from a character to its ASCII value with `ord(ch)`. You can convert the other way from an ascii value to a char with `chr(num)`. For example, in the python interpreter you could type:

```
>>> print(ord('e'))
>>> print(ord('r'))
>>> print(ord('r') - ord('e'))
>>> print(chr(97))
>>> print(chr(68))
## now look at the difference between digits and their ascii values
>>> print(chr(51))
>>> print(chr(51) - chr(49))
```

34.2 Weak crypto

We will look at two types of weak cyphers: *Caesar* and *substitution* cyphers. We will write programs that encode text with these cyphers, and then we will look at how to write programs to attack the cyphers and possibly decrypt messages.

34.2.1 A simple Caesar encryptor

In a Caesar code we shift each letter in the alphabet by a given number. If the shift is 5 then 'a' becomes 'f', 'l' becomes 'q', 'x' becomes 'c' and so forth. The world 'hello' rotated by 5 becomes 'mjqqt', and 'hello' rotated by 13 becomes 'uryyb'.

This program will rotate its input by 13 characters. Our program will shift the ascii values and then produce the corresponding characters. Type the program `caesar.py` in Listing **??** into a file `caesar.py`.

Listing 34.2.1: caesar.py – simple Caesar cypher: rotate the characters in the input 13.

```
#! /usr/bin/env python3
import sys

def main():
    shift = 13
    encrypted_line = ''
    fname = ''
    if len(sys.argv) > 2:
        print('error: usage is "%s [filepath]"' % sys.argv[0])
        sys.exit(1)
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        fname = 'standard-input'
        f = sys.stdin
    ## now go through the file, one character at a time
    for c in f.read():
        if c < 'a' or c > 'z':  # we only handle 'a' <= c <= 'z'
            encrypted_line = encrypted_line + c # add it with no encryption
            continue
        ascii_value = ord(c)
        ## find the new character by shifting this character
        crypt_char = ascii_value + 13
        if crypt_char > ord('z'):
            ## if we go beyond z we wrap back around to 'a'
            crypt_char = (crypt_char - ord('a')) % 26 + ord('a')
        # print(c, '->', crypt_char)
        encrypted_line = encrypted_line + chr(crypt_char)
    print('encrypted: ', encrypted_line)

main()
```

Exercise 34.1 A very simple exercise: change `caesar.py` to take an argument which is the shift number for the Caesar cipher, that way it does not have to be 13. You would see if `sys.argv[1]` is set, and if so use `int(sys.argv[1])` as your offset.

Exercise 34.2 As a slightly more elaborate exercise you could take `caesar.py` and adapt it to handle upper case characters as well as lower case ones.

34.2.2 Substitution ciphers

A simple substitution makes a table between clear text letters and encrypted letters. They don't have have to be shifted by a fixed amount: the shift can be different for each character, but each clear text character will always map to the same secret encrypted letter.

Here is an example of substitution in which 'a' is mapped to 'b', 'b' to 'o', 'c' to 'm', 'd' to 'l', and so forth up to 'z' being mapped to 'p'. There seems to be no rhyme or reason to the mapping. The word 'hello' woulud become 'itxxn':

```
abcdefghijklmnopqrstuvwxyz
bomltzhiqjgxwdnsvackeuyfrp
```

Note that the substitute characters do not have to be letters of the alphabet, as long as there is a bijective mapping (one-to-one correspondence) between the substitute characters and the letters of the alphabet.

A program which encrypts with that substitution cypher is `substitution.py` in Listing **??**.

Listing 34.2.2: substitution.py – simple substitution cypher: replace each character with its substitute.

```python
#! /usr/bin/env python3
import sys

def main():
    original   = list('abcdefghijklmnopqrstuvwxyz')
    #substitute = list('bomltzhiqjgxwdnsvackeuyfrp')
    substitute = list('omltzniqjgxwdhsvackeuyfrpb')

    encrypted_line = ''
    fname = ''
    if len(sys.argv) > 2:
        print('error: usage is "%s [filepath]"' % sys.argv[0])
        sys.exit(1)
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        fname = 'standard-input'
        f = sys.stdin
    ## now go through the file, one character at a time
    for c in f.read():
        if c < 'a' or c > 'z':  # we only handle 'a' <= c <= 'z'
            encrypted_line = encrypted_line + c
            continue
        ## pos is the position in the original list
        pos = ord(c) - ord('a')
        ## find the new character by taking that position in the
        ## substitute list
        crypt_char = substitute[pos]
```

(continues on next page)

(continued from previous page)

```
        # print(c, '->', crypt_char)
        encrypted_line = encrypted_line + crypt_char
    print('encrypted: ', encrypted_line)

main()
```

34.2.3 A "literary" substitution cypher

Edgar Allan Poe wrote a short story called "The Gold-Bug" (see https://www.gutenberg.org/files/2147/2147-h/2147-h.htm#link2H_4_0006 and https://en.wikipedia.org/wiki/The_Gold-Bug and http://self.gutenberg.org/articles/eng/The_Gold-Bug)

In this story a Mr. Legrand is obsessed with decrypting a cryptogram purportedly written by the famous pirate Captain Kidd, giving directions to his mythical buried treasure. The cryptogram reads:

```
53‡‡†305))6*;4826)4‡.)4‡);806*;48†8
¶60))85;;]8*;:‡*8†83(88)5*†;46(;88*96
*?;8)*‡(;485);5*†2:*‡(;4956*2(5*-4)8
¶8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡
1;48†85;4)485†528806*81(‡9;48;(88;4
(‡?34;48)4‡;161;:188;‡?;
```

34.2.4 Preparing to attack substitution cyphers: frequency analysis

The field of *cryptanalysis* is more than a thousand years old. Most classical cyphers were substitution cyphers, and the method of *frequency analysis* is effective in attacking those.

To see how this works, let us remember the words of Mr. Legrand in Edgar Allan Poe's "The Gold-Bug":

> "Now, in English, the letter which most frequently occurs is e. Afterwards, succession runs thus: a o i d h n r s t u y c f g l m w b k p q x z. E predominates so remarkably that an individual sentence of any length is rarely seen, in which it is not the prevailing character.
>
> "Here, then, we leave, in the very beginning, the groundwork for something more than a mere guess. The general use which may be made of the table is obvious [. . .]

The key words are "[. . .] groundwork for something more than a mere guess": the happiest words for a code breaker to hear.

Let us make a *histogram plot* of the frequency with which letters occur in English. How do we make such a plot? And how do we get a statistically valid sample of English text so that we can count how often letters typically occur?

To make the plot we can write the program `letter_frequency.py` in Listing **??**.

Listing 34.2.3: letter_frequency.py – calculate and plot the frequency of each letter.

```
#! /usr/bin/env python3

import sys

def main():
    fname = ''
```

(continues on next page)

(continued from previous page)

```
    if len(sys.argv) > 2:
        print('error: usage is "%s [filepath]"' % sys.argv[0])
        sys.exit(1)
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        fname = 'standard-input'
        f = sys.stdin

    letter_count = [0]*26
    for c in f.read():
        if not c.isalpha(): # special cases: character is not alphabetic
            continue
        if ord(c) > 127:        # or not ascii
            continue
        ## special case: character is uppercase, so we fold it down to
        ## lowercase
        if c.isupper():
            c = c.lower()
        ## now we have a lower chase char, so we find its position in
        ## the alphabet
        pos = ord(c) - ord('a')
        ## now increase the letter_count histogram for that position
        letter_count[pos] += 1
    f.close()
    print_frequency_histogram(fname, letter_count, offset=ord('a'))

def print_frequency_histogram(title, hist, offset=0):
    print('====== frequency histogram for %s ======' % title)
    for i in range(len(hist)):  # simple ascii table+histogram
        c = chr(i+offset)
        if not (i + offset in range(ord('A'), 1+ord('Z'))
                or i + offset in range(ord('a'), 1+ord('z'))
                or i + offset in range(ord('0'), 1+ord('9'))):
            c = ' '
        print('%3d -- %s %8d --  '
              % (i, c, hist[i]), end="")
              # % (i, chr(i + ord('a')), hist[i]), end="")
        ## now print a histogram of the letter X, but scale it down so
        ## the max has some 50 Xs
        n_Xs_to_print = float(hist[i]) * 50.0 / max(hist)
        for j in range(int(n_Xs_to_print)):
            print('X', end="")
        print()

if __name__ == '__main__':
    main()
```

Now let us analyze three different samples of English words with our `letter_frequency.py` program.

/usr/share/dict/words

The easiest is probably to start with /usr/share/dict/words since this dictionary is already present in every GNU/Linux system. We will write a program which reads in the file, counts all the letters, and then puts out a simple list of how much they occur.

You can see the output of this program on /usr/share/dict/words:

```
====== frequency histogram for /usr/share/dict/words ======
  0 -- a    67956 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  1 -- b    16446 --  XXXXXXXX
  2 -- c    33242 --  XXXXXXXXXXXXXXXXX
  3 -- d    29683 --  XXXXXXXXXXXXXXX
  4 -- e    92097 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 -- f    11146 --  XXXXX
  6 -- g    23682 --  XXXXXXXXXXXX
  7 -- h    20490 --  XXXXXXXXXX
  8 -- i    69461 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  9 -- j     2080 --  X
 10 -- k     9057 --  XXXX
 11 -- l    43064 --  XXXXXXXXXXXXXXXXXXXXXX
 12 -- m    23656 --  XXXXXXXXXXXX
 13 -- n    59577 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 14 -- o    51269 --  XXXXXXXXXXXXXXXXXXXXXXXXXX
 15 -- p    23100 --  XXXXXXXXXXXX
 16 -- q     1604 --
 17 -- r    59717 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 18 -- s    95874 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 19 -- t    54763 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXX
 20 -- u    27214 --  XXXXXXXXXXXXXX
 21 -- v     8436 --  XXXX
 22 -- w     8002 --  XXXX
 23 -- x     2312 --  X
 24 -- y    13164 --  XXXXXX
 25 -- z     3478 --  X
```

You will notice something strange: the letter ‘s’ seems to occur about as much as the letter ‘e’, or even a bit more. This goes against what Mr. Legrand had stated. An exercise below will address this issue.

A work by Shakespeare

King Lear by William Shakespeare can be found at http://www.gutenberg.org/ebooks/1128 and the plain text version is at: http://www.gutenberg.org/cache/epub/1128/pg1128.txt

We can download its text and analyze it with:

```
wget --output-document king-lear.txt http://www.gutenberg.org/cache/epub/1128/pg1128.txt
python3 ./letter_frequency.py king-lear.txt
```

```
...
```

```
====== frequency histogram for king-lear.txt ======
  0 -- a     8373 --  XXXXXXXXXXXXXXXXXXXXXXXXX
```

(continues on next page)

(continued from previous page)

```
  1 -- b     1734 --  XXXXX
  2 -- c     2602 --  XXXXXXXX
  3 -- d     4519 --  XXXXXXXXXXXXX
  4 -- e    16159 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 -- f     2512 --  XXXXXXX
  6 -- g     2580 --  XXXXXXX
  7 -- h     7046 --  XXXXXXXXXXXXXXXXXXXXX
  8 -- i     7300 --  XXXXXXXXXXXXXXXXXXXXXX
  9 -- j       89 --
 10 -- k     1290 --  XXX
 11 -- l     5150 --  XXXXXXXXXXXXXXX
 12 -- m     3244 --  XXXXXXXXXX
 13 -- n     7317 --  XXXXXXXXXXXXXXXXXXXXXX
 14 -- o     9751 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 15 -- p     1841 --  XXXXX
 16 -- q       78 --
 17 -- r     7703 --  XXXXXXXXXXXXXXXXXXXXXXX
 18 -- s     7150 --  XXXXXXXXXXXXXXXXXXXXXX
 19 -- t    10796 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 20 -- u     4494 --  XXXXXXXXXXXXX
 21 -- v      481 --  X
 22 -- w     2726 --  XXXXXXXX
 23 -- x      120 --
 24 -- y     2794 --  XXXXXXXX
 25 -- z       34 --
```

A contemporary English book

Looking at the books on Project Gutenberg sorted by release date at

https://www.gutenberg.org/ebooks/search/?sort_order=release_date

we can pick “An Ocean Tragedy” by William Clark Russell and download its text and then run our program with:

```
wget --output-document ocean-tragedy.txt https://www.gutenberg.org/files/56363/56363-0.
↪txt
python3 ./letter_frequency.py ocean-tragedy.txt
```

```
...
```

```
====== frequency histogram for ocean-tragedy.txt ======
  0 -- a    64744 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  1 -- b    12390 --  XXXXXX
  2 -- c    18785 --  XXXXXXXXX
  3 -- d    34787 --  XXXXXXXXXXXXXXXXXX
  4 -- e    95677 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 -- f    20728 --  XXXXXXXXXX
  6 -- g    18775 --  XXXXXXXXX
  7 -- h    53974 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXX
  8 -- i    59395 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  9 -- j      973 --
```

(continues on next page)

(continued from previous page)

```
10 -- k     7401 --  XXX
11 -- l    34322 --  XXXXXXXXXXXXXXXXX
12 -- m    19202 --  XXXXXXXXXX
13 -- n    54177 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXX
14 -- o    60199 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
15 -- p    13126 --  XXXXXX
16 -- q      820 --
17 -- r    44082 --  XXXXXXXXXXXXXXXXXXXXXXX
18 -- s    51953 --  XXXXXXXXXXXXXXXXXXXXXXXXXXX
19 -- t    71954 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
20 -- u    22794 --  XXXXXXXXXXX
21 -- v     6639 --  XXX
22 -- w    21089 --  XXXXXXXXXXX
23 -- x     1308 --
24 -- y    14615 --  XXXXXXX
25 -- z      586 --
```

Conclusion: signatures

Take a close look at the three histograms, the one from `/usr/share/dict/words` has a certain feeling

Exercises

Exercise 34.3 Discuss with your fellow students why the histogram of letters in `/usr/share/dict/words` does not resemble that in actual books.

Exercise 34.4 Are there significant differences in the letter frequency between Shakespeare and the modern book? Discuss this with your fellow students.

Exercise 34.5 Study the differnce in letter frequency distributions between short and long bits of text. Take a few single-paragraph files and look at their letter distribution and compare it to what you got with a full book.

Exercise 34.6 Look at books in other languages and see what the letter distribution is there. Can you identify the language a book was written in just by looking at the distribution of letters? Note: the program `letter_frequency.py` will run on a european language file that has accented letters (like Italian or German or French) which are not part of ASCII, but it will do so by dropping all those characters. You will want to adapt it to handle those special accented characters.

34.2.5 Applying the frequency analysis to a message

Let's do this as a sequence of exercises:

Exercise 34.7 First type a reasonably long message in English into a file. Then run your `caesar.py` or `substitution.py` on that file to encrypt it and save the output to another file.

Exercise 34.8 Email that file to a "friend" and to a hypothetical "foe". Only tell your friend what the Caesar shift or the substitution map is, and see if the foe can crack the code.

Exercise 34.9 Alternate being writer, friend and foe so that everyone in your group gets to decrypt a message.

Exercise 34.10 Is it easier to decode short or long messages?

Exercise 34.11 The *foe* should try to crack the code without knowing what the substitution table is. How can she use the `letter_frequency.py` program to do this?

34.3 Strong crypto

We have seen how easy it is to break simple substitution cyphers. Those patterns in the distribution of letters are a clear sign that we are not encrypting very intelligently.

Between the 1880s and 1910s stronger methods of encryption were invented. The *one time pad* is a theoretically unbreakable method of encryption. Both sender and recipient have the same random sequence of symbols, and each character they send gets folded in some way with the next symbol in the one time pad.

We will implement something similar to the one time pad, but first we need to review some aspects of binary representation of numbers.

34.3.1 Binary numbers, XOR, hiding the byte

Remember that characters are represented by their ASCII codes. The ASCII code for ‘g’ is 103 in decimal, 0x67 in hex, and 01100111 in binary.

We want to find a way to *fold* ‘g’ with some random byte (remember: a byte is an 8-bit number). Let’s choose 01101010. We could fold the two together in many different ways, but we want to then be able to separate them back out.

The approach that is usually used is to use the XOR operator. What is this?

We will apply the various logic operators to bits: if 0 means false and 1 means true, then we have the following truth tables:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

NOT 0 = 1
NOT 1 = 0
```

Sometimes people find the behavior of OR to be counter-intuitive because “1 OR 1 = 1”. People sometimes remember their parents saying “you can have cake OR you can have ice cream but not both”. That violates “1 OR 1 = 1”, but parents don’t usually accept this logic. For parents and cryptographers we have another logical operation called XOR (*exclusive* OR). The truth table for XOR is:

```
0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0
```

This was done one bit at a time. Suppose you have the character ‘g’ (01100111) and our pattern 01101010. The XOR of those two is done bit-by-bit:

```
01100111  ('g')
  XOR
01101010  (number from one time pad)
   |
```

(continues on next page)

(continued from previous page)

```
   v
00001101  (encrypted number)
```

The cool thing about the XOR function is that if you XOR with the same number again you get back the original: all those bits get flipped back:

```
01100111 XOR 01101010 -> 00001101 (the encrypted number)
00001101 XOR 01101010 -> 01100111 (back to 'g'!)
```

What does this mean? It means that if you know the secret number you can encrypt your numbers, and you can also decrypt them! And someone who does not have have the secret number cannot make any progress.

34.3.2 Manipulating bits in python

Python offers us the "bitwise xor" operator which uses the symbol ^

Let us see examples of how to use it:

```
## in the python interpreter, at the >>> prompt, type the following
x = int('01100111', 2)
y = int('01101010', 2)
z = x ^ y
print('%d ^ %d = %d' % (x, y, z))
print('0x%x ^ 0x%x = 0x%x' % (x, y, z))
print('%s ^ %s = %s' % (bin(x), bin(y), bin(z)))
print('to restore the original number:')
print('%s ^ %s = %s' % (bin(x), bin(z), bin(z^y)))
print(bin(x), bin(z^y), x == z^y)
```

34.3.3 Revisiting random number generators

We have explored random numbers before, but our angle here is slightly different. We are now interested in how two different people can get the *same* stream of random numbers.

First let us look at an example of how to get the first ten numbers in a stream. At the Python interpreter type:

```
import random
for i in range(10):
     print(i, random.randint(0, 255))
```

Now run the same thing in a different terminal. You will see that the two streams are quite different, even though you are using the same random number generator in both examples.

The reason for the difference is that the *starting point* for the sequence is different. This starting point is called the random number seed, and by default it is set to a value determined by some real time event, like the current wall clock time.

But almost all random number generators allow you to *set* the seed to a given value, so now let us try running the following code in two different terminals:

```
import random
random.seed(7419)
```

(continues on next page)

(continued from previous page)

```
for i in range(10):
    print(i, random.randint(0, 255))
```

The output will be the same for both: when we set an identical seed, the random sequences are identical.

Note that the seed (which I set to 7419) could be any integer – each different seed will produce a different sequence, but the if you use that integer twice then you get the same sequence.

This setting of the random number seed will be crucial to allowing a person to decrypt our message.

34.3.4 Implementing tougher encryption

The idea is be to generate a stream of random numbers, use them to encrypt a stream of bytes with the XOR operation. Later we will use that same sequence to decrypt the encrypted bytes.

The program which will encrypt text is `pad_encrypt.py` in Listing **??**.

Listing 34.3.1: pad_encrypt.py – Simple encrypting program which uses a one-time pad (in our case a sequence of random numbers) to encrypt a file.

```
#! /usr/bin/env python3
import sys
import random

def main():
    ## we need a known sequence, so we use a known random number seed
    seed = 1234
    random.seed(seed)
    fname = ''
    if len(sys.argv) > 2:
        print('error: usage is "%s [filepath]"' % sys.argv[0])
        sys.exit(1)
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        fname = 'standard-input'
        f = sys.stdin
    ## now process the file
    encrypted_text = ''
    for c in f.read():
        # print(c, ord(c), bin(ord(c)))
        ascii_value = ord(c)     # turn char into an integer
        if ascii_value > 255:
            continue
        next_random = random.randint(0, 255) # get next random number
        ## find the new character by XORing it with the random number
        crypt_char = ascii_value ^ next_random
        ## add the encrypted character to the stream of encrypted text
        encrypted_text = encrypted_text + chr(crypt_char)
    dump_bytes_as_hex(encrypted_text)
```

(continues on next page)

(continued from previous page)

```
def dump_bytes_as_hex(bytes, show_ascii=False):
    """print out a stream of bytes as 2-digit hex bytes; optionally
    also show if there are valid ASCII substrings"""
    hex_line = ''
    ascii_line = ''
    for i in range(len(bytes)):
        if i > 0 and i % 15 == 0:
            print_line(hex_line, ascii_line, show_ascii)
            hex_line = ''
            ascii_line = ''
        hex_line += ' 0x%02x' % ord(bytes[i])
        if bytes[i].isalnum():
            ascii_line += bytes[i]
        else:
            ascii_line += '.'
    ## print the final bytes
    print_line(hex_line, ascii_line, show_ascii)

def print_line(hex_line, ascii_line, show_ascii):
    print(hex_line, end="")
    if show_ascii:
        ## attempt at being "clever": print 4 spaces plus some extra
        ## if the line is shorter, so that the printable part is
        ## always aligned
        print(' '*(4 + 5*(15 - len(ascii_line))), end="")
        print(ascii_line, end="")
    print()

main()
```

You can look at the result of encrypting a small file by typing a paragraph into a text file and running `pad_encrypt.py filename` and for very short snippets you can do something like `echo "just three words" | python3 pad_encrypt.py` and get:

```
0x8b 0x4e 0x70 0x5a 0x31 0x5e 0x5a 0xc7 0x1c 0x6d 0x2f 0x7f 0xde 0x85 0x89
0x3f 0x24
```

As you can see the program `pad_encrypt.py` outputs the encrypted bytes as hex numbers that look like 0x5a. This is because these encrypted bytes would seldom be readable as part of a string: most of them would not be ascii values for anything printable.

34.3.5 Decrypting this tougher encryption

The program to decrypt text encoded by `pad_encrypt.py` is `pad_decrypt.py` in Listing **??**.

Listing 34.3.2: pad_decrypt.py – Simple decrypting program which uses a one-time pad (in our case a sequence of random numbers) to encrypt a file.

```
#! /usr/bin/env python3
import sys
import random
```

(continues on next page)

(continued from previous page)

```
def main():
    ## we need a known sequence, so we use a known random number seed
    seed = 1234
    random.seed(seed)
    fname = ''
    if len(sys.argv) > 2:
        print('error: usage is "%s [filepath]"' % sys.argv[0])
        sys.exit(1)
    elif len(sys.argv) == 2:
        fname = sys.argv[1]
        f = open(fname, 'r')
    else:
        fname = 'standard-input'
        f = sys.stdin
    encrypted_letter_count = [0]*256
    decrypted_letter_count = [0]*26
    decrypted_text = ''
    for line in f.readlines():
        words = line.split()
        ## extract the encrypted characters
        for word in words:
            crypt_char = int(word, 16)
            encrypted_letter_count[crypt_char] += 1 # track a histogram
            next_random = random.randint(0, 255) # get next random number
            ascii_value = crypt_char ^ next_random
            if ascii_value >= ord('a') and ascii_value <= ord('z'):
                decrypted_letter_count[ascii_value-ord('a')] += 1 # track a histogram
            decrypted_char = chr(ascii_value)
            ## add the decrypted character to the stream of decrypted text
            decrypted_text = decrypted_text + decrypted_char
    from letter_frequency import print_frequency_histogram
    print_frequency_histogram('king lear encrypted',
                              encrypted_letter_count)
    print()
    print_frequency_histogram('king lear decrypted',
                              decrypted_letter_count, offset=ord('a'))
    ## now write the output to a file ending in _decrypted
    fout = fname + '_decrypted'
    open(fout, 'w').write(decrypted_text)
    print('output written to %s' % fout)

def dump_bytes_as_hex(bytes, show_ascii=False):
    """print out a stream of bytes as 2-digit hex bytes; optionally
    also show if there are valid ASCII substrings"""
    hex_line = ''
    ascii_line = ''
    for i in range(len(bytes)):
        if i > 0 and i % 15 == 0:
            print_stuff(hex_line, ascii_line, show_ascii)
            hex_line = ''
            ascii_line = ''
```

(continues on next page)

(continued from previous page)

```
        hex_line += ' 0x%02x' % ord(bytes[i])
        if bytes[i].isalnum():
            ascii_line += bytes[i]
        else:
            ascii_line += '.'
    ## print the final bytes
    print_stuff(hex_line, ascii_line, show_ascii)

def print_stuff(hex_line, ascii_line, show_ascii):
    print(hex_line, end="")
    if show_ascii:
        print(' '*(4 + 5*(15 - len(ascii_line))), end="")
        print(ascii_line, end="")
    print()

if __name__ == '__main__':
    main()
```

You will notice that `pad_decrypt.py` has an interesting feature: it uses the function `print_frequency_histogram()` from `letter_frequency.py` to show us the frequency histogram for both the encrypted and decrypted text:

```
...
```

```
====== frequency histogram for king lear encrypted ======
  0 --        590 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  1 --        581 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  2 --        615 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  3 --        586 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  4 --        654 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 --        624 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  6 --        620 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  7 --        607 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  8 --        614 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  9 --        596 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 10 --        614 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
...
249 --        596 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
250 --        598 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
251 --        669 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
252 --        630 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
253 --        569 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
254 --        620 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
255 --        587 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

====== frequency histogram for king lear decrypted ======
  0 -- a     7882 --  XXXXXXXXXXXXXXXXXXXXXXXXX
  1 -- b     1345 --  XXXX
  2 -- c     2220 --  XXXXXXX
  3 -- d     4274 --  XXXXXXXXXXXXX
  4 -- e    15707 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
  5 -- f     2050 --  XXXXXX
```

(continues on next page)

(continued from previous page)

```
  6 -- g     2064 --  XXXXXX
  7 -- h     6733 --  XXXXXXXXXXXXXXXXXXXXX
  8 -- i     6345 --  XXXXXXXXXXXXXXXXXXXX
  9 -- j       85 --
 10 -- k     1039 --  XXX
 11 -- l     4586 --  XXXXXXXXXXXXXX
 12 -- m     2948 --  XXXXXXXXX
 13 -- n     7042 --  XXXXXXXXXXXXXXXXXXXXXX
 14 -- o     9503 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 15 -- p     1602 --  XXXXX
 16 -- q       61 --
 17 -- r     7450 --  XXXXXXXXXXXXXXXXXXXXXXX
 18 -- s     6516 --  XXXXXXXXXXXXXXXXXXXX
 19 -- t    10001 --  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 20 -- u     4442 --  XXXXXXXXXXXXXX
 21 -- v      414 --  X
 22 -- w     2320 --  XXXXXXX
 23 -- x      118 --
 24 -- y     2663 --  XXXXXXXX
 25 -- z       31 --
output written to king-lear-encrypted-pad.txt_decrypted
```

You should be delighted at how the encrypted text has no discernible pattern in its frequency distribution, while the decrypted version has the pattern we are familiar with.

This means that one *cannot* decrypt this code using frequency analysis.

34.4 Further reading

34.4.1 Technical details

- https://en.wikipedia.org/wiki/XOR_cipher

34.4.2 Historical

- https://en.wikipedia.org/wiki/Venona_project which contains the statement "[. . .] Due to a serious blunder on the part of the Soviets, some of this traffic was vulnerable to cryptanalysis. The Soviet company that manufactured the one-time pads produced around 35,000 pages of duplicate key numbers, as a result of pressures brought about by the German advance on Moscow during World War II. [. . .]"
- "The Code Book", by Simon Singh

34.4.3 Videos

- Crash Course https://www.youtube.com/watch?v=jhXCTbFnK8o
- Cris Moore’s talk on cryptography: https://www.youtube.com/watch?v=3J2RHGVf0ho

CHAPTER

THIRTYFIVE

OTHER LANGUAGES - GO

Section author: Sophia Mulholland <smulholland505gmail.com>

https://golang.org/dl/

35.1 Hello World in Go

Go is similar in syntax to C, yet simpler in many ways. C definitely has more structure and syntax rules that Go does not have, which makes Go simpler and easier to learn. Writing a short program in Go looks like a combination of C and Python. Syntax like this allows you to get comfortable with the language quickly and move on to working on harder things.

First, we will get comfortable with running a Go program. In an editor such as emacs, open up a new file called "hello.go".

The first program to write in Go is a simple "Hello World" which is shown below:

Listing 35.1.1: 'hello.go'

```
package main

import "fmt"

func main() {
        fmt.Println("Hello, World!")
}
```

Notice: Every Go program should have a '**package main**' defined at the top of the code, with the majority of programs using that package. The line '**import "fmt"**' is basically like writing '#include <stdio.h>' in C. It imports the package 'fmt' that allows you to print things. The last thing to notice is that Go does not have any semicolons.

Compile and run this code with

```
$ go build hello.go
$ ./hello
$ Hello, World!
```

OR optionally the Go compiler supports a 'go run' command which will just compile and run it in the same step:

```
$ go run hello.go
$ Hello, World!
```

35.2 Writing a Go program with command line arguments

Please see the chapter on differential equations for the context to this program. It is meant to graph the output of using the euler method to solve equations and compare it with the exact answer.

Let's write another simple program to get familiar with the syntax of the language called 'euler-method.go'.

Listing 35.2.1: 'euler-method.go' solves differential equation using Euler's method

```go
package main

import (
        "fmt"
        "math"
        "os"
        "strconv"
)

func differential (x float64, y float64) float64 {
        return (2*x)-2
}

func solution (sx float64, sy float64) float64 {
        return math.Pow(sx, 2) - (2*sx)
}

func main() {
        var n_steps int = 1
        if len(os.Args) == 1 {
                n_steps = 1000
        } else if len(os.Args) == 2 {
                n_steps, _ = strconv.Atoi(os.Args[1])
        } else {
                fmt.Println("error usage is %s [n_steps]\n", os.Args[0])
                os.Exit(1)
        }
        var set_y float64 = 0
        // define initial x, initial y, step size, and # of steps
        var interval float64 = 10
        var dt float64 = interval/float64(n_steps)
        var xi float64 = 0
        var yi float64 = set_y
        for j := 0; j < int(n_steps); j++ {
                //find exact solution to compare approximation with
                var exact float64 = solution(xi, yi)
                //solve differential with (xi, yi)
                var m float64 = differential(xi, yi)
                //use tangent line to approximate next y value from previous y value
                var new_y float64 = yi + dt * m
                //increase x by step size
                var new_x float64 = xi + dt
                xi = new_x
```

(continues on next page)

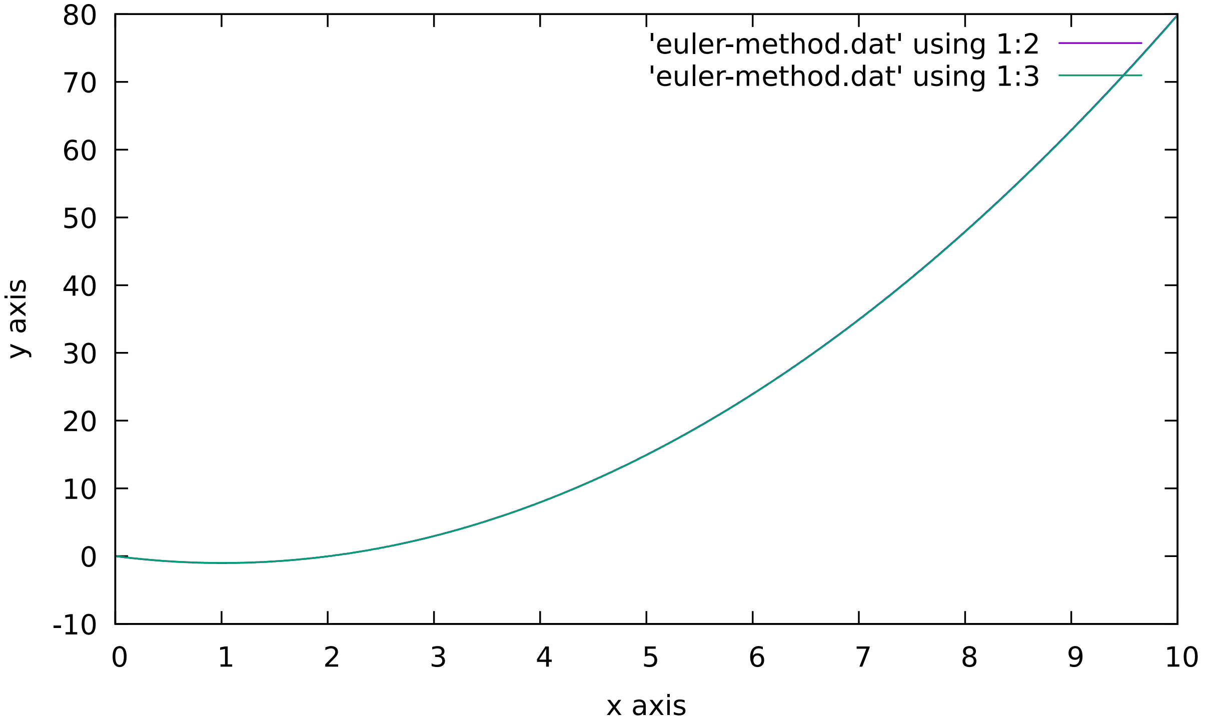
(continued from previous page)

```
                yi = new_y
                fmt.Println(xi, yi, exact)
        }
}
```

As we can see in the code, Go has a different way of declaring variables and handling command line arguments. In Go, handling command line arguments is simpler as you can import the package “os” and not have to deal with the argc and *argv[] or understanding pointers yet. By just importing “os” you can deal with command line inputs as an array and can check the length of that array rather than dealing with the number argc.

To deal with converting this array of string inputs into integers, we can import “strconv” which converts the strings into ints. However, this strconv function converts it into a number plus a <nil> at the end so avoid this compiler error when we try to declare n_steps, we can declare the variable n_steps along with the blank identifier ‘_’ to avoid having to declare all the return values (<nil>).

```
$ go build euler-method.go
$ ./euler-method 1000 > euler-method.dat

gnuplot> plot "euler-method.dat" using 1:2 with lines
gnuplot> replot "euler-method.dat" using 1:3 with lines
```



35.3 Goroutines and channels

A cool thing that Go does well is run **goroutines**. These are functions that can run concurrently with other functions. **Concurrency** means that the functions are running at the same time rather than waiting for one to finish and then the next program starts. Basically it is used when we need to handle multiple things at the same time. Go makes these kinds of programs simple to write and easier to understand because concurrency is generally difficult to understand in computing. Also, these goroutines do not take up a lot of memory and so they are very fast.

Threads are most commonly used to run things concurrently. All processes have at least one main thread that executes a task. Threads take a fixed amount of memory in the stack to create and when many threads are used, switching between them gets messy.

However, using goroutines is like a step above threads. The programmer deals with them while the computer does not even know goroutines exist. They take little memory to create, 2kB, which means you can have millions of them running on one CPU. They can also shift in size, to accommodate the stack. Go also deals with the switching of goroutines for the programmer, pausing and running. This is an advantage to goroutines because the programmer does not need to say ahead of time when to stop and start the threads.

Let's see how to implement these goroutines.

Listing 35.3.1: 'goroutines-hello.go'

```go
package main

import (
        "fmt"
)

func hello() {
        fmt.Println("Hello world")
}

func main() {
        go hello()
        fmt.Println("main function")
}
```

We have a function called hello, which looks just like a normal declaration of a function and what makes it a goroutine is putting 'go' in front of the function call. So let's run it.

```
$ go build goroutines-hello.go
$ ./goroutines-hello
```

The output is

```
$ main function
```

That's weird. We called the function hello(), and it did not print "Hello world" like we wanted. That is because by making it a goroutine, the compiler did NOT wait for it to finish its task of printing "Hello world" and it went on with the rest of the main function. When the main function finishes, the program is over no matter what, and hello() did not have enough time to finish.

Let's write a program that deals with more goroutines and introduce a way to make sure the program waits for the goroutines to finish.

Listing 35.3.2: 'goroutines.go'

```go
package main

import (
        "fmt"
        "time"
)

func hello() {
        fmt.Println("Hello World")
}

func listnumbers() {
        for i := 0; i < 10; i++ {
                fmt.Println(i)
        }
}

func squareroot(x int) {
        if x == 2 {
                fmt.Println("Oh no, a 2")
        } else {
                fmt.Println("good choice")
        }
}

func main() {
        //run all three goroutines
        go hello()
        go listnumbers()
        go squareroot(3)
        //wait for the goroutines to finish
        time.Sleep(2 * time.Second)
        fmt.Println("All finished")
}
```

```
$ go build goroutines.go && ./goroutines
```

If we ran it in the exact same way, the goroutines would not have time to finish so let's help it do everything it's supposed to. Notice how we added a line for the computer to "sleep" as the goroutines finished and it we run it, it prints out everything we wanted it to. If you run it multiple times you'll find the order of the outputs changes, as depending on which goroutine is running on which core, it will finish first.

So these goroutines are cleaner and easier to implement than in C where you need pointers and lots of lines of code where it can get messy quickly.

So, in these programs we have no way of knowing when the goroutines end. Therefore, we have to make the program wait for two seconds to be sure it does. This is where **channels** come in.

A way goroutines can communicate with each other is through channels. Channels can send and receive data of one type. If a goroutine sends something to a channel, it automatically waits until a goroutine can receive that data. It will be blocked until the goroutine requests that data and vice versa.

Listing 35.3.3: 'channels-example.go'

```go
//channels-example.go
package main

import (
        "fmt"
        "time"
)


func pinger (c chan string) {
        for i := 0; ; i++ {
                //send "ping" to the channel
                c <- "ping"
        }
}
//channel gets "ping" and waits until printer() is ready to receive "ping"

func printer (c chan string) {
        for {
                //receive "ping" from channel and assign it to msg
                msg := <- c
                fmt.Println(msg)
                time.Sleep(time.Second)
        }
}

func main() {
        //makes a channel called c
        //strings are passed on it
        var c chan string = make(chan string)

        go pinger(c)
        go printer(c)

        //input receives data from the channel c
        //input := <- c
        //will print ping
        //fmt.Println(input)

        var input string
        //scans the input from c channel and prints it out
        //does not stop until you exit the program running
        fmt.Scanln(&input)
}
```

```
$ go build channels-example.go && ./channels-example
$ ping
```

So this program created the channel c, and ran two goroutines (pinger() and printer()). Pinger() sends "ping" to the channel and Printer() prints out the message it received from the channel. In the main function, we also show that the variable 'input' can receive data from the channel too and print it out.

So goroutines allow functions to be run concurrently and channels are how these goroutines communicate. Go offers an easy way to implement them and it is

35.4 More complicated Go program

Lastly, let's take a look at a more complicated program from the differential equations chapter rewritten in Go.

Listing 35.4.1: 'damped-spring-with-friction-plain.go'

```
package main

import (
        "fmt"
        "math"
)

// acceleration due to gravity
var g = 9.8
// physics values for spring equation
var k = 3.0
var m = 2.0
// physics for air friction
var air_friction = 3
// physics for frictional damping force
var damp = 0.5
var F0 = 10.0                        // constant driving force


func Acceleration (t float64, v float64, x float64) float64  {
        return ((-k*x) - (damp*v) + (F0*math.Cos(8.0*t))) / m
}

func Harmonic_exact (t float64, st float64, sx float64, sv float64) float64 {
        //return math.Cos(t)
        return sx*math.Cos(math.Sqrt(k/m - damp*damp/(4*m*m))*t)*math.Exp((-damp/(2*m))* t)
}

func Falling_body_exact (t float64, st float64, sx float64, sv float64) float64 {
        return sx + sv*t - 0.5*g*math.Pow(t,2)
}

func Damped_spring () {
        //set variables for falling_body_exact()
        var set_t float64 = 0
        var set_x float64 = 5   /* try 10 for falling body, 5 for harmonic */
        var set_v float64 = 0        /* try 10 for falling body, 0 for harmonic */
        //initial variables
        var interval float64 = 0.001 // how many seconds
        var n_steps float64 = 1000000
        var dt = interval/n_steps // size of steps
        var ti = set_t
```

(continues on next page)

(continued from previous page)

```
        var xi = set_x
        var vi = set_v
        for j := 0; j < int(n_steps); j++ {
                var exact float64 = Harmonic_exact(ti, set_t, set_x, set_v)
                var acc float64 = Acceleration(ti, vi, xi)
                var new_v = vi + (dt * acc)
                var new_x = xi + (dt * vi)
                // increase t by dt
                var new_t = ti + dt

                vi = new_v
                xi = new_x
                ti = new_t
                fmt.Println(ti, vi, xi, exact);
        }
}

func main () {
        Damped_spring();
}
```

So this program outputs a solution to a differential equation solved by Euler's method. The length is pretty much the same as the C program however you can see there are no double types in Go. Instead, it offers the equivalent float64 type which indicates it requires 64 bits in memory.

```
$ go build damped-spring-with-friction-plain.go
$ ./damped-spring-with-friction-tut > damped-spring.dat
gnuplot> plot “damped-spring.dat” using 1:3 with lines
gnuplot> replot “damped-spring.dat” using 1:4 with line
```

This output is the same as the chapter on differential equations, go to that chapter for an explanation on what it means. That chapter has the same program as this damped spring one, except it is written in C.

SOURCES

https://golang.org/cmd/go/

https://github.com/vladimirvivien/go-cshared-examples

https://golangbot.com/hello-world/

https://medium.com/rungo/achieving-concurrency-in-go-3f84cbf870ca

https://www.sohamkamani.com/blog/2017/08/24/golang-channels-explained/

CHAPTER

THIRTYSIX

APPENDIX: AN ITINERARY FOR GUEST LECTURES

This appendix shows some examples of possible guest lectures which would show some of the material in this book. It is mostly meant to provide links

36.1 Motivation for linking computing and scholarship

36.1.1 Personal story

- 1980 and the first personal computres
- 1983 and the Reed College physics department requiring that we learn to program
- 1983 and David Basin trading: he teaches me C and UNIX if I teach him to play guitar
- 1985 and Nerdix
- 1985 and watching Keith Packard hack
- Realizaing that others did not get this level of computing instruction
- 1992 and an impossible physics thesis thanks to software sophistication

36.1.2 Software freedom

The only reason I have enjoyed my career as a physicist is because of the existence of software freedom.

36.1.3 What can be done in various disciplines

Although I have not seen it represented graphically, there is a timeline

36.1.4 How a physicist-hacker looks at things

The glow worms in New Zeland.

Constellations in the random number calculation of pi.

36.2 A tour of topics

36.2.1 The history and calculation of pi

Section **??**

36.2.2 Generating music

To give this presentation it is useful to first have the students install the sox utilities on their own computers.

Section **??**

- https://www.dataquest.io/blog/free-datasets-for-projects/

CHAPTER

THIRTYSEVEN

APPENDIX: HOW TO BUILD THE BOOK

37.1 Motivation, prerequisites, plan

Our goal here is to show you how to build the book from its raw text into the HTML and other formats in which the book is published.

We use the Sphinx typesetting system bla bla python docs use Sphinx bla bla FIXME

Prerequisites:

- Installing the tools needed to clone and build the book.
- Learning how to use the Sphinx documentation system.

37.2 The tools needed

We need tools to get the book from the network repository (a git repository, hosted on codeberg.org), and to build the book (using the sphinx documentation system).

```
$ sudo apt install -y git make wget
$ sudo apt install -y gnuplot-qt
$ sudo apt install -y ffmpeg
$ sudo apt install -y python3-pip
## we also need to run the examples that make plots for the books;
## for that we need scipy and matplotlib
$ python3 -m pip install --user --upgrade matplotlib numpy scipy
## install the LaTeX packages needed to build the printable book
## with "make latexpdf"
$ sudo apt install -y texlive-latex-base texlive-latex-recommended
$ sudo apt install -y texlive-fonts-recommended
$ sudo apt install -y biber latexmk graphviz dot2tex
$ sudo apt install -y librsvg2-bin pdf2svg
$ python3 -m pip install --user --upgrade sphinx sphinxcontrib-bibtex sphinxcontrib-
↪jsmath
$ python3 -m pip install --user --upgrade sphinxcontrib-programoutput
$ python3 -m pip install --user --upgrade sphinx-rtd-theme

$ python3 -m pip install --user --upgrade sphinxcontrib.programoutput
$ python3 -m pip install --user --upgrade sphinxcontrib.bibtex
$ python3 -m pip install --user --upgrade recommonmark
$ python3 -m pip install --user --upgrade sphinxcontrib.bibtex
```

37.3 Version control: cloning the repository (you only do this once)

We use git to access the book repository on the `https://codeberg.org/` version control site.

If you go to https://codeberg.org/markgalassi/serious-programming-courses you will find instructions to clone the repository with git. They will look something like:

```
## cloning instruction if you have an account on codeberg:
$ git clone git@codeberg.org:markgalassi/serious-programming-courses.git
## cloning instruction if you are coming in anonymously
$ git clone https://codeberg.org/markgalassi/serious-programming-courses.git
```

The first link will work for anyone, but then you cannot push changes back. The second is for collaborators who are modifying the book.

You can then change in to the directory into which you have cloned the repository, and examine what you have, with:

```
$ cd serious-programming-courses
$ ls
$ cd small-courses
$ ls
$ find . -name '*.rst'
```

That last `find` command will show you all the files that end in `.rst`. These files contain the text of the book written as simple text with the light-weight *markdown* annotations. Examine the files.

37.4 Building the book

The sphinx documentation system will build the book in either HTML or EPUB format. The details are all taken care of and you can do so by typing:

```
$ cd serious-programming-courses/small-courses
$ make html           ## (or "make epub")
```

You can then point your browser to the file `_build/html/index.html` to see what it all looks like.

37.5 Making and committing changes (your day-to-day)

You can make changes to the `.rst` files. When you do so you need to re-run `make html` and then view the changes by reloading the HTML files in your browser.

First you should incorporate changes made by other people. You type:

```
$ git pull
```

Then you can make some changes to files, after which you can *commit* them to the git repository with

```
$ git commit -a
```

This commit will open an editor so that you can put in a log message. This message should be thoughtful and describe briefly what changes you made to each file you modified.

If you are a collaborator on the book and have write permission on sourceforge you can now push your changes for other collaborators to see with:

```
$ git push
```

(If you do not have write permission you can contact the authors to get your changes to them.)

So the *workflow* is:

- Pull and update from sourceforge.
- Make changes to the book by modifying the `.rst` files.
- Commit the changes with `git commit -a`
- If you have write permission on codeberg, push in the code with `git push`

CHAPTER

THIRTYEIGHT

APPENDIX: HOW TO ADD A CHAPTER

Motivation

The purpose of this appendix is to show you how you might contribute a chapter to this book. You should pay attention to the *structure* of how the topic is presented: what sections do we have at the beginning and end of the chapter, how we introduce exercises, and then be able to start writing about your topic.

Prerequisites

- An idea you want to write about.
- Learning how to use the Sphinx documentation system to add to the book. This is described in Section **??**.

Plan

In keeping with our hands-on approach I will demonstrate the idea with a sample chapter that is actually an interesting mini-course.

The topic I will use for this demonstration is the *quadratic formula*. This is an important topic: one of the few formulas that you learn in middle school and then might go on using for the rest of your life.

38.1 Anatomy of the chapter

1. Motivation, prerequisites, plan
2. Simplest example of a calculation.
3. Simplest example of a plot.
4. A short Python program which generates or analyzes data related to the topic.
5. Applications to real-world situations, such as science or "life".
6. Exercises. Exercises can be inserted at any point in the text. The exercises can also lead in to more advanced work.
7. Further study. References and ideas to learn more about the topic than can be done in the 1.5 hour mini-course.

I will now show what the sample chapter looks like, prefixing every section title with "The chapter:" and putting notes in square brackets explaining the "pedagogical" motivation for some of what I put into the chapter.

NOTE: from here on we show the mock chapter.

38.2 The chapter: Title

Title: The quadratic formula

38.3 The chapter: frontmatter

Motivation

Our goal here is to learn to find *roots of second degree polynomials*. What does this mean? If we have a second degree polynomial, which looks like $ax^2 + bx + c$, we look for values of x that give the result $ax^2 + bx + c = 0$.

We will work toward the *quadratic formula*. This, like the *Pythagoras theorem*, is one of the few formulas which many people continue to use in life after they leave school. This is because second degree polynomials come up in many areas of science, even areas so simple that we might deal with them in life.

Our examples will involve:

physics
: figuring out how long it takes a ball that you drop to hit the ground

geometry
: or what is the biggest area rectangle you can draw with a fixed perimeter.

Prerequisites

- The 10-hour “serious programming” course.
- The “Data files and first plots” mini-course in Section **??**

Plan

We will start by introducing the equation to be solved, then we will show the solution, after which we will look at some simple examples and then some examples of scientific calculations we can make with the quadratic formula.

For some very simple polynomials we will see that we can *guess* the solution, but most of the time it will not be that easy, so we will show:

- The quadratic formula, which finds the roots to second degree polynomials (Section **??**).
- A *numerical approximation*. We will write a computer program which scans through the real number line looking for zeros (Section **??**). This technique also applies to more complicated functions than the quadratic equation.

38.4 The chapter: The problem

38.4.1 A basic equation

The problem is that, given numbers a, b and c, we want to find a value of x for which:

$$ax^2 + bx + c = 0 \tag{38.4.1}$$

For example, if $a = 1$ and $b = 2$ and $c = 0$ then we have a simple case of Equation (**??**):

$$x^2 - 5x + 6 = 0 \tag{38.4.2}$$

and a bit of guesswork tells us that both $x = 2$ and $x = 3$ will make Equation (**??**) out:

Plugging $x = 3$ into Equation (**??**) we get:

$$3^2 - 5 \times 3 + 6 = 9 - 15 + 6 = 4 - 4 = 0$$

And plugging $x = 0$ into Equation (**??**) we get:

$$2^2 - 5 \times 2 + 6 = 4 - 10 + 6 = 0$$

so Equation (**??**) is satisfied when you plug 2 into x, and also when you plug 3 into x.

For this polynomial we were able to find the roots by guesswork. As I mentioned, most of the time we will not be able to guess the solutions.

38.4.2 Some terminology

Polynomial
A function $f(x)$ which is a sum of terms with x at various powers. For example $f(x) = 7x^3 + 2x^2 - 4x + 2$

Second degree polynomial
A polynomial where the highest power is x^2.

Parabola
The shape of the plot of a second degree polynomial.

Roots of a polynomial
The values for x for which $y = 0$. These are the *solutions* to the equations we have been looking at.

Numerical approximation
An approximate solution to a problem found by carrying out calculations that get you closer and closer to a solution.

Numerical analysis
The academic discipline which resolves around finding numerical approximations to the solutions of mathematical problems.

38.4.3 But wait! Two solutions??

[Pedagogical goal: I want to toss in a mnemonic which has always helped me in understanding solutions to equations. A first order equation has up to one solution; a second order equation has up to two solutions, and so forth.]

You should find it notable that there are two solutions to this equation. You might have previously studied equations like $7x + 4 = 0$ which have a single solution $x = -4/7$. So why do we have two solutions for this one?

Here are at least three ways of looking at this:

Squaring numbers removes the minus sign

If you take a simple second degree equation like $x^2 = 16$, you can easily see that $x = 4$ and $x = -4$ will both give you 16 when you square x. So the existence of the term x^2 means that you can have up to two different values of x that satisfy the same equation.

Multiplying two first degree polynomials

Returning to Equation (**??**), let's take a look at the two solutions 2 and 3. Now form two first degree equations that are easily solved by 2 and 3: $x - 2$ and $x - 3$. If one of those expressions is always zero when x is a solution, then the product of them $(x - 2) \times (x - 3)$ will always be zero. So we can write:

$$(x - 2) \times (x - 3) = 0$$

But we can also work out that product of $(x - 2) \times (x - 3)$:

$$(x - 2) \times (x - 3) = x^2 - 2x - 3x + 2 \times 3 = x^2 - 5x + 6$$

which is our euqation again! This is another way to see clearly that both settings for x, 2 and 3, will make that equation be zero.

Plotting the polynomial and looking at zero crossings

If you plot the polynomial, as we will see below in Figure **??** and other figures in that section, you see that the shape of the plot is what we call a *parabola*, and that the values of x for which it crosses the x axis also depend on the values of a, b and c.

38.5 The chapter: Plots

A very energetic math professor in college used to always say "Let's get a picture going!" when embarking on understanding something new.

So here are some plots to get a visual feel for what these *solutions* are. We start with the simplest polynomial $y = x^2$ which has a single solution at $x = 0$.

Listing 38.5.1: Simplest second degree polynomial $y = x^2$

```
set grid
plot x**2
```

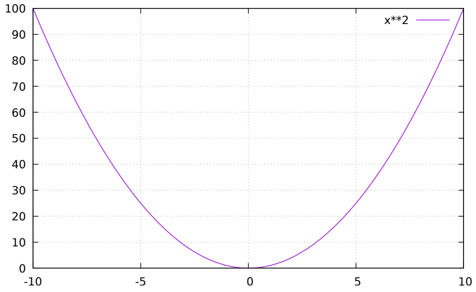


Figure 38.5.1: Plot of the simplest second degree polynomial $y = x^2$. Note the only root is at $x = 0$.

Then we look at a more complicated second degree polynomial. The shape is the mostly similar, but the position is different. You can see from Figure **??** that it cuts through the x axis in two points: $x = 2$ and $x = 3$.

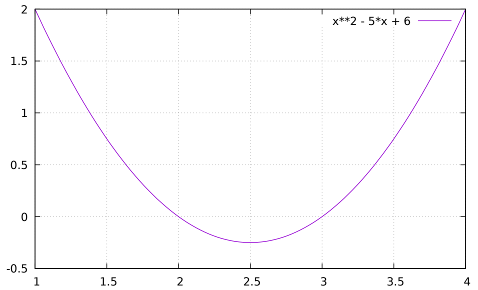


Figure 38.5.2: Plot of the second degree polynomial $y = x^2 - 5x$. Note the roots at $x = 2$ and $x = 3$. Also note that we had to narrow the values of x to go between 1 and 4 (instead of the default -10 to 10 that gnuplot does). This is so that we can see more closely where $y = 0$ occurs.

Listing 38.5.2: Second degree polynomial $y = x^2 - 5x + 6$

```
set grid
plot [1:4] x**2 - 5*x + 6
```

And now for an example where you have *no* solutions: y will never be zero.

Listing 38.5.3: Second degree polynomial $y = x^2 - x + 24$

```
set grid
plot x**2 - x + 24
```

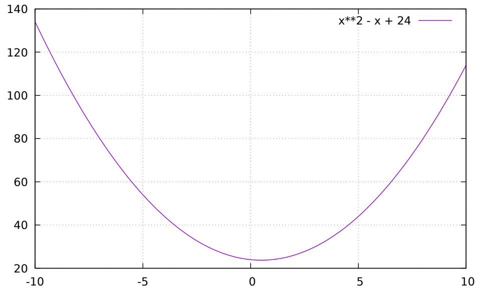


Figure 38.5.3: Plot of the simplest second degree polynomial $y = x^2 - x + 24$. Note that there are no roots: the plot is entirely above the x axis.

Our final plot will show the example of a plot which points downward. This happens when the x^2 term has a minus sign on it.

Listing 38.5.4: Second degree polynomial $y = -x^2 + x + 2$

```
set grid
plot [-3:4] -x**2 + x + 2
```

What do we get out of all these plots? We have explored the landscape of second degree polynomials and seen that changing the parameters a, b and c affects the direction and placement of the figure, including the places in which the plot crosses the x axis. These are called the *roots* of the polynomial.

We have also seen how the values of a, b and c determine if there will be 2 solutions (the two arms of the parabola intersect the x axis), or one solution (the bottom of the parabola grazes the x axis), or no solutions (the parabola is

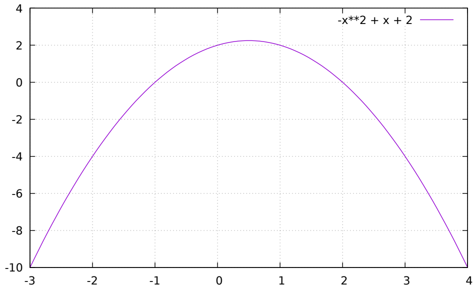


Figure 38.5.4: Plot of the second degree polynomial $y = -x^2 + x + 2$ which points the opposite way: the minus sign in $-x^2$ makes the branches of the parabola point down instead of up. It has roots at $x = 1$ and $x = -2$.

entirely above or entirely below the x axis).

38.6 The chapter: The quadratic formula

Here is our centerpiece: the quadratic formula. The two solutions to the equation

$$ax^2 + bx + c = 0$$

are:

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

As we mentioned above, there can be up to two solutions. Sometimes there will be one solution (when $b^2 - 4ac = 0$), and sometimes there will be *no* real solutions (when $b^2 - 4ac < 0$, since as you know the square root of a negative number is not a real number).

So how do we apply this? Here are some examples from the plots we showed in Section **??**.

In the equation $x^2 = 0$ we have $a = 1, b = 0, c = 0$ so there is a single root:

$$x = \frac{-0 \pm \sqrt{0^2 - 4 \times 1 \times 0}}{2 \times 1} = 0$$

Looking at $y = x^2 - 5x + 6$ we have $a = 1, b = -5, c = 6$, so the two roots are:

$$x = \frac{-(-5) \pm \sqrt{(-5)^2 - 4 \times 1 \times 6}}{2 \times 1} = \frac{5 \pm \sqrt{25 - 24}}{2} = \frac{5 \pm 1}{2} = (3, 2)$$

Now for the polynomial $y = x^2 - x + 24$. This has $a = 1, b = -1, c = 24$ and the quadratic formula gives:

$$x = \frac{-(-1) \pm \sqrt{(-1)^2 - 4 \times 1 \times 24}}{2 \times 1} = \frac{1 \pm \sqrt{-95}}{2} = \frac{1 \pm \sqrt{-95}}{2}$$

which has no solutions because $b^2 - 4ac = -95$ which is negative and thus has no real square roots.

Finally the upside-down parabola $y = -x^2 + x + 2$ has $a = -1, b = 1, c = 2$ and the solutions are:

$$x = \frac{-1 \pm \sqrt{1^2 - 4 \times (-1) \times 2}}{2 \times (-1)} = \frac{-1 \pm \sqrt{5}}{-2} = ((1 - \sqrt{5})/2, (1 + \sqrt{5})/2)$$

All of these solutions should match the places in the plots where the graph crosses the axes.

38.7 The chapter: Numerical approximation

[Pedagogical goal: it's important to have a bit of programming for each math/science topic we demonstrate.]

We have two goals in this section: one is to demonstrate how you can write programs to give a *numerical* approximation to the correct answer.

The other goal is to show a method which will also help with more complicated root-finding problems. You see, quadratic equations are one of the simplest kinds of equations. Once you go to cubic (third degree) polynomials the formula is much more complicated, for quartic (fourth degree) polynomials it's crazy, and for fifth degree and higher there are no formulae.[1]

"Root finding" is one of the most studied topics in numerical analysis and we will just scratch the surface here, but we will get some good results. Type in the program in Listing **??** and try running it.

Listing 38.7.1: numerical-zeros.py Look at where a second degree polynomial crosses zero.

```
#! /usr/bin/env python

"""This program shows an example of looking for roots of a polynomial.
We start with a guess that the roots will be between x = -10 and x= 10
and sample points until we find that y has changed sign.
"""

import math

## FIXME: must write it; look for zero crossings by looking for a
## change in sign

lowest = -10
highest = 10
step = 0.01

def main():
    x = lowest
    y = poly(x)
    previous_sign = -1 if y < 0 else 0
    while x <= highest:
        y = poly(x)
        sign = -1 if y < 0 else 1
        x += step                 # move along the x axis
        if previous_sign != sign:
            print('we found a root at approximately x = %g' % x)
        previous_sign = sign

def poly(x):
    y = -x**2 + x + 2
    # y = x**2 - 5*x + 6
    # y = 7*x**2 + 2*x - 40
    ## now try a higher order polynomial
    # y = 2*x**5 - 3*x**4 - x**3 + 2*x**2 + x - 1
```

(continues on next page)

[1] The proof that there is no general formula for fifth degree and higher polynomials is one of the deepest and most beautiful theorems in mathematics; it is the subject of *Galois Theory*. Sadly it is too complex to describe in this footnote.

(continued from previous page)

```
    return y

main()
```

It will tell you an approximate

38.8 The chapter: Applications

38.8.1 Physics: falling bodies

One of the earliest successes of physics, hundreds of years ago, was being able to calculate what happens with falling bodies, both those that fall straight to the ground and those that are constrained (like a pendulum).

The force pulling a body to the ground on the surface of the earth is given by:

$$F = -mg$$

where m is the object's mass (typically measured in kilograms, *kg*), and g is the *acceleration of gravity* on the earth's surface (typically measured in meters per seconds squared, m/s^2).

You can then couple this with Newton's second law for how any body feels a force:

$$F = ma$$

where a is the acceleration produced by that force. With some elementary calculus you can find what the height of the body is at any moment in time $h(t)$. If you have not yet studied calculus then bear with me and you will soon see the result which you can just trust.

The acceleration is the second derivative of the position: $a = \frac{d^2h(t)}{dt^2}$. The solution to the physics equation is:

$$h(t) = h_0 + v_0 t - \frac{1}{2} g t^2$$

where:

- h_0 is the height from which we drop it. Let's say we hold it up to 2 meters.
- v_0 is the initial velocity you give it. In our case $v_0 = 0$ because we are just letting it drop.

So now our physics question is: after how much time does the body hit the ground?

To answer this we note that we hit the ground when the height is 0: $h(t) = 0$. If we substitute the physics equation for $h(t)$ we get:

$$0 = h_0 + v_0 t - \frac{1}{2} g t^2$$

Since v_0 is 0, we have:

$$h_0 - \frac{1}{2} g t^2 = 0 \tag{38.8.1}$$

If we put in the values we discussed for initial height and acceleration of gravity ($h_0 = 2$ and $g = 9.81$) we get:

$$-4.905 \times t^2 + 2 = 0$$

Now we can use our quadratic formula with $a = -4.905, b = 0, c = 2$ and we get:

$$t = \frac{-0 \pm \sqrt{0^2 - 4 \times (-4.905) \times 2}}{2 \times -4.905} = \frac{\pm\sqrt{39.24}}{-9.81} = \pm 0.64 \tag{38.8.2}$$

The two solutions are then ± 0.64 which is measured in seconds.

In a physics problem, when you have two possible solutions due to a "plus or minus" in the square root you look at which of those times makes sense physically. Clearly the negative solution (it hits the ground in the past) does not make sense, so we conclude that the time we hit the ground is:

$$t = 0.64 \text{ seconds}$$

This value of t solves the equation for when the object hits the ground.

Exercises for this section

1. Try dropping a somewhat dense object from a height of 2 meters and see if you can get a good measurement of how long it takes to hit the ground.
2. Solve the same equation for t using a height of 3 meters and 4 meters, and see if you can *safely* time how long it takes a body to fall from those heights.
3. Write out the general solution to Equation (**??**). You can use the quadratic formula with $a = -g, b = v_0, c = h_0$.
4. Discuss how to pose a more interesting problem in which you give the ball a slight upwards toss. This basically boils down to using a small initial velocity v_0. Then take the more general form of the equation that you just wrote out in the previous exercise and plug in your value for v_0 as well as those for h_0, g, and calculate the time at which the object will hit the ground.
5. I mentioned that the solution for t has two values, due to the $\pm$ that comes from solving a second degree polynomial. I told you to ignore the negative time solution, but try discussing with your classmates what the meaning of that negative time solution might be if you started tracking the trajectory of your body *before* the moment at which you drop it.

38.8.2 Geometry: areas and the Dido problem

[Pedagogical note: I like to introduce associated historical facts to make the subject more fun, to connect it to other subjects, and to remind students that all areas of knowledge shine light on each other.]

Now we solve a geometrical problem, inspired by the legend of Dido, queen of Carthage.

The backstory is that some 8 or more centuries BCE, Dido led a party of Phoenicians to settle an area in what is today Tunisia. She negotiated permission with a local Berber king to found her colony in an area big enough to be encircled by an ox hide.

She cut the hide into very narrow strips, and thus had a fixed length of hide and needed to draw the shape with the biggest area that could be surrounded by that hide.

So "Dido's problem" is: what shape gives you the most area with a fixed perimeter?

The general mathematical solution of this problem is hard to prove, but we will compare two candidate shapes with a fixed perimeter: a square and a circle. We will then calculate the areas of each and see which would be a better choice.

If we call the perimeter p, then we have:

$$\begin{aligned} A_c &= \pi r^2 \\ p &= 2\pi r \end{aligned}$$

which gives

$$A_c = \frac{\pi p^2}{(2\pi)^2} = \frac{p^2}{4\pi}$$

For the square:

$$A_{sq} = r^2$$
$$p = 2\pi r$$

which gives:

$$A_{sq} = \frac{p^2}{(2\pi)^2} = \frac{p^2}{4\pi^2}$$

FIXME: must still find the killer app for a quadratic formula on this.

38.9 Exercises for the chapter

[Pedagogical goal: apart from the usefulness of doing exercises, these are formulated to draw students beyond the material they should be prepared for – sometimes even beyond what is reached in high school, like 3rd degree polynomials.]

Third degree polynomial equations like $ax^3 + bx^2 + cx + d = 0$ have messy-looking solutions (and higher degree polynomals even more so), so we will not look in to them here, but some of these exercises will show some of the cleaner cases that can come up.

Exercise 38.1 On paper expand the expression $(x - 3)(x - 2)(x - 1)(2x - 1)(x + 1)$ into a polynomial. It should be a fifth degree polynomial. With your classmates find a way to plot it so that you see the ranges well (hint: I came up with setting an x range of *[-2:4]* and a y range of *[-6:14]*). Can you guess what the roots are? Does the plot sem to confirm your guess? Discuss with your classmates why the plot diverges so much at the left and right ends.

Exercise 38.2 Repeat the steps in the previous exercise, but this time craft your own *sixth* degree polynomial. Discuss how the plot is different from the previous one.

Exercise 38.3 Make up a seventh degree polynomial, plot it, then put that polynomial in the program in Listing **??** to find its roots. Do the roots found in your program seem to match what you see in your plot?

Exercise 38.4 Look for the roots of non-polynomial equations. For example: the $\sin(x)$ function has roots at $(0, \pi, 2\pi, 3\pi, ...)$ and in fact at any integer multiple of π. Can you find approximations to that?

Exercise 38.5 The program in Listing **??** has a rather coarse step size of 0.01. Discuss how this limits the accuracy of your solution (i.e. by how much have your solutions been "off" of the correct solution?), and try making it smaller to see if the accuracy gets better. Try making it *really* small (how about 1.0e-6, which is scientific notation for one millionth) and see if the program starts taking too long to execute.

38.10 Further study

https://en.wikipedia.org/wiki/Quadratic_formula

CHAPTER

THIRTYNINE

APPENDIX: PROJECT PROPOSALS

Contents of this chapter

Motivation

This appendix has some ideas for student projects. They are crafted with the idea of getting students to:

- craft or using interesting algorithms
- write some code
- explore a topic from a wide selection of areas of endeavor
- optionally then contribute a chapter to this book

Each section of this appendix should ideally have a link to a concise and clear definition of the topic, and ideas on how to start the exploration.

39.1 Social sciences

Note that some of the biology topics, like phylogenetic analysis, are also social science areas.

39.1.1 Optimal stopping and life/business

Subject areas: psychology, economics

One variant of optimal stopping is the Marriage Problem, or Secretary Problem. It is described at https://www.geeksforgeeks.org/secretary-problem-optimal-stopping-problem/

Possible project: craft a simulation modeling the applicant pool as a list of N applicant scores. Then generate a very large number of applicant pools.

For each applicant pool go through them and stop after k applicants, taking note of the best score so far. Then continue, picking the first applicant with a score greater than that "best so far". Record this for all possible k (between 1 and N), and show which stopping point k would be the best.

Compare this with the theoretical value of 1/e.

Possible references:

- Brian Christian and Tom Griffiths - "Algorithms to live by: the computer science of human decisions". The chapter on "Optimal stopping: when to stop looking".
- Blog post inspired by the book: https://medium.com/geekculture/computer-science-algorithms-in-daily-life-optimal-stopping-608

39.1.2 Multi-armed bandits and exploration vs. exploitation

Subject areas: psychology, economics

Look at the problem of exploration vs. exploitation in general, and then consider human and societal situations in which it can be applied.

Psychology: risk aversion in different phases of life.

Management: employee tasking in different phases of a project.

Project: using literature from psychology, find data sets that show how real people do exploration vs. exploitation, and compare them to the theoretical result.

Possible references:

- Brian Christian and Tom Griffiths - "Algorithms to live by: the computer science of human decisions". The chapter on "Explore/exploit : the latest vs. the greatest"
- https://medium.com/geekculture/computer-science-algorithms-in-daily-life-explore-vs-exploit-a613232cf9e1

39.1.3 Deadly conflicts

Subject areas: history.

Explore data sets (which can be found in the references below) on the statistics of casualties in war. Look for a pedagogically useful visualization of one or more data sets.

The two plots that could be made involved (a) the power law showing the general behavior, and (b) trend plots throughout history.

A discussion of power laws can then be interesting.

Possible references:

- "The Better Angels of Our Nature: Why Violence Has Declined" - by Steven Pinker. Chapters: The statistics of deadly quarrels, Part 1 : the timing of wars ; The statistics of deadly quarrels, Part 2 : the magnitude of wars.
- Aaron Clauset: "Trends and fluctuations in the severity of interstate wars" - https://www.science.org/doi/10.1126/sciadv.aao3580
- Cunen, Hjort, Nygard: "Statistical sightings of better angels: Analysing the distribution of battle-deaths in interstate conflict over time". Article at: https://journals.sagepub.com/doi/10.1177/0022343319896843 and replication data at: https://www.prio.org/journals/jpr/replicationdata

39.1.4 Quantifying overfitting in personal decisions

[. . .]

39.1.5 Just about anything from Gwern Branwen

Gwern Branwent's web site has a vast collection of research interests, mostly related to social science. It would be good to distill projects from them.

https://www.gwern.net/

https://www.gwern.net/Archiving-URLs

39.2 Sports

39.2.1 Time series for improvement on records

Start out by having a pure poisson process to generate improvement in "world records". For example, try something like:

```
import numpy as np
s = np.random.poisson(5, 10000)
```

as shown at https://numpy.org/doc/stable/reference/random/generated/numpy.random.poisson.html

Keep track of when a new sample beats all previous records, and study the time between improvements on world records.

Then:

Examine the history of improvement in individual world records, for example in the 100m swimming (both female and male), 100m running, high jump, … Compare to the pure Poisson distribution.

In chess examine the history of surpassing "highest ever" Elo ratings for (a) top player, (b) top 10 average, (c) top 100 average, and look at when previous records get broken. Try the same with alternative chess metrics.

Try to see if this hypothesis is valid: improvements in world records are randomly distributed (maybe Poisson? how about other distributions?) But sometimes you have a jump sooner than expected by the statistics. This could correspond to a "freak of nature" player (but the statistics could also generate such players. More interesting are advances in training technique and in equipment. Also of note might be an increase in the population that plays that sport.

39.2.2 Optimal tournament structure

Create a group of (say) 32 virtual tournament players, and assign them a rating, similar to chess Elo ratings.

Then set them up in double-round-robin, round-robin, swiss system, and direct knockout.

Examine the consistency of the results.

Ask couple of questions (and maybe more):

- What is the best tournament format for various situations?
- How many places can be determined (just 1st? 1st, 2nd, 3rd? …) using the double-round-robin as the gold standard.

39.2.3 Soccer analytics

See what you can do starting from this article about Messi in the 538 blog:

https://fivethirtyeight.com/features/lionel-messi-is-impossible/

39.3 Physical sciences

39.3.1 Brownian motion

See if it is straightforward to write an agent-based model (or other simulation) for brownian motion.

Then see if Einstein's 1905 results on Brownian motion can be derived from this simulation.

Discuss, with these results in hand, how all of this relates to the existence of atoms.

https://en.wikipedia.org/wiki/Brownian_motion

39.4 Life sciences

39.4.1 Datasets for phylogenetic analysis

Subject areas: biology, ecology, linguistics

Find specific datasets and create examples of phylogenetic trees.

These are more likely to come from biology, but they could also come from linguistics or other areas of social science.

Start with canned examples, like in the 2021 Medium article by Rishika Gupta, and learn to make phylogenetic trees (including going beyond her examples).

Then look for coded datasets from language or mythology or other social science sources.

Possible references:

- https://biopython.org/wiki/Phylo
- http://etetoolkit.org/
- https://medium.com/geekculture/phylogenetic-trees-implement-in-python-3f9df96c0c32
- https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-021-04274-6

39.4.2 Predator-prey ecology: equations and agents

Subject areas: biology, ecology

Craft a predator-prey scenario in two different ways: (a) by solving the Lotka-Volterra equations as shown in Almond Heil's model Section **??**, and (b) using the agent-based modeling system shown in Section **??**

Start by adapting Almond Heil's model to predator-prey interaction (you might also get this from the basic Mesa examples).

Then try to set up the same model in a Lotka-Volterra setting: solve the Lotka-Volterra model with a differential equation solver, and see if its cyclic population behavior matches what you get from the agent-based model.

After looking at data and plots, do some thinking about the assumptions that go in to both kinds of models, and try to make them match more closely.

If you cannot get a close match, do some thinking on the mechanisms behind both approaches and try to explain the differences.

Possible references:

- Section **??**
- https://mesa.readthedocs.io/en/latest/
- https://towardsdatascience.com/introduction-to-mesa-agent-based-modeling-in-python-bcb0596e1c9a
- https://github.com/projectmesa/mesa

39.4.3 Infectious disease modeling

Subject areas: biology, epidemiology

The basic model used for infectious diseases is the SIR model, where you track "susceptible", "infected", and "removed" population members.

You can also model it with agent-based models.

So the idea here is to look at the SIR model in two different ways: (a) using the agent-based modeling system shown in Section **??**, and (b) by solving the SIR differential equations.

Start by adapting Almond Heil's model to have variables that track the full S, I, and R populations, and color them appropriately.

Then set up the SIR differential equations to have the same starting values, and run them with a simple differential equation solver - you can write your own Euler's method solver, or use the scipy runge-kutta solver.

Possible references:

- Section **??**
- https://www.maa.org/press/periodicals/loci/joma/the-sir-model-for-spread-of-disease-the-differential-equation-model
- https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology
- https://www.frontiersin.org/articles/10.3389/fams.2020.571544/full
- https://www.niss.org/sites/default/files/SIR_Modeling_tutorial_ob.pdf
- https://mesa.readthedocs.io/en/latest/
- https://towardsdatascience.com/introduction-to-mesa-agent-based-modeling-in-python-bcb0596e1c9a
- https://github.com/projectmesa/mesa

39.5 Mathematics

39.5.1 puzzles

[must flesh this out]

Jason Breshears at https://archaics.com/ has a youtube video where he presents what's special for the number 2178 in his observation.

39.5.2 On the role of intuition in mathematics

Read in depth this article on mathematical intuition by famous mathematician Terence Tao:

https://terrytao.wordpress.com/career-advice/theres-more-to-mathematics-than-rigour-and-proofs/

and try to devise a simple problem in mathematics (maybe one of the ones below) that could illustrate the path from pre-rigorous to rigorous to post-rigorous stages in one's mathematical journey.

39.5.3 Monty Hall's door problem

Write a simulation of the famous Monty Hall door problem. The simulation should be a very simple programming task.

This could be done with a language you already know, or as an exercise to learn a new programming language, since the simulation will be very simple.

- Leonard Mlodinow "The Drunkard's Walk", chapter 3 "Finding Your Way Through a Space of Possibilities"
- https://www.youtube.com/watch?v=AD6eJlbFa2I
- https://www.youtube.com/watch?v=4Lb-6rxZxx0

39.5.4 Random walks

Subject areas: mathematics

Look at the theory of random walks and compare it to simulations.

Possible references: the chapter on randomness and disorder, Section **??**

39.5.5 Runge-kutta method

Subject areas: mathematics

Compare the runge-kutta library (gsl or scipy) result to a hand-coded Euler method. Compare first-point, mid-point, and last-point of Euler's method to runge-kutta.

The compare them all to the exact solution for the nonlinear pendulum, and show the difference.

39.5.6 Sync and the Kuramoto model

Subject areas: mathematics

Look at the simple mathematical models for sync described in Steven Strogatz's lecture:

https://youtu.be/RpU7JrE1uCk

and implement the simple mathematical models that he shows.

He discusses the Kuramoto model at about 40 minutes into the video, and he shows an example of a program that visualizes that model.

Learn the model, and then figure out how to implement it in python, starting with text output and then using a simple widget set for the visualization.

Possible references:

- https://youtu.be/RpU7JrE1uCk

- https://en.wikipedia.org/wiki/Kuramoto_model
- https://www.complexity-explorables.org/explorables/ride-my-kuramotocycle/
- http://www.complexity-explorables.org/slides/ride-my-kuramotocycle/
- https://www.ted.com/talks/steven_strogatz_the_science_of_sync

39.5.7 Analysis of chess

No articulation of a project yet, but the ideas are approximately:

- How does the advantage of the white pieces change with player ratings, beginner to grandmaster? And to time controls?
- What are the standard deviations of victory predictions based on ratings?

Possible sources:

- lichess.org API
- https://content.iospress.com/articles/icga-journal/icg0012
- https://en.wikipedia.org/wiki/Comparison_of_top_chess_players_throughout_history
- https://www.quora.com/Has-Stockfish-or-any-other-top-chess-program-performed-an-analysis-of-the-Karpov-Kasparov-matche
- https://lichess.org/blog/YafSBxEAACIAr0ZA/exact-exacting-who-is-the-most-accurate-world-champion
- Understanding distributions of chess performances. Study and reproduce this paper by Regan:

https://cse.buffalo.edu/~regan/papers/pdf/

https://cse.buffalo.edu/~regan/papers/pdf/RMH11.pdf

(There is also an RMH11b.pdf – which is the more recent version?)

Also look at this quora answer that mentions it: https://www.quora.com/What-is-the-most-important-move-in-chess/answer/H%C3%A5kon-Hapnes-Strand?ch=10&oid=340561749&share=7fabf913&srid=3MvD&target_type=answer

39.6 Computer science and information technology

39.6.1 Situational awareness of your network

Subject areas: IT, computer programming

The idea here is to understand what devices are on your network, and what services they offer.

Start by familiarizing yourself with the commands in the "How to get insight into a network" chapter of the Sysadmin Hacks book at at https://markgalassi.codeberg.page/sysadmin-hacks-html/ (you might want to look at the previous chapter "The basics of networking" for background).

In particular the sections on the *nmap* and *pnscan* commands are the starting point for what we want to do. Observe the command line output for each of them with the examples shown in the book.

Then separately install the program *etherape* and look at what it does.

Finally, look at the graphical output from the program *map_network_utils.py* in that chapter.

At this point you can take two different directions: one is to keep using separate utilities to visualize the network, in this case the powerful *graphviz* tool. Graphviz allows a lot of very fancy layout options, and quite a bit more information about the network can be added to the display in the form of logos for the operating system (Linux, android, . . .), or for the protocol (ssh, imap, . . .), or one could try to probe traffic information and visualize that.

- https://graphviz.org/gallery/

39.7 Humanities and the arts

39.7.1 Music generation: tone beyond sin waves

Topics: music

Take the simple sine wave generators shown in Section **??** and investigate how to make the tone more like that of a guitar string.

This will involve moving from a *sin()* wave to a more complex funciton. One start could be to add the harmonics to make a sawtooth wave or a triangle wave, but you should do some research to find out what simple tone generators sound interesting.

39.7.2 Music generation: add stereo

Topics: music

Take the simple sine wave generators shown in Section **??** and investigate how to add an artificial sterephonic feeling.

Right now the examples generate identical left and right channel signals. The idea here is to look at how to modify them slightly in various ways, and to see if that creates a stereo effect.

After a bit of tinkering and experimentation you could then do research to see if there is any known mathematics for doing this, and implement those techniques.

39.7.3 Further explorations into Zipf's law

Subject areas: literature, digital humanities

Start from the simple programs that demonstrate Zipf's law in Section **??**. Then read Isabella Trejo's presentation from the Institute for Computing in Research at https://github.com/izzytrejo/Zipf to understand some of the proposed underpinnings for Zipf's law in literary text.

Then investigate some mathematical properties of the various translations, such as the power law slopes and the power law breaks.

Does this happen consistently as you look at the translation of different books into the same other languages?

How does this depend on the translator? For example, can you find examples of the same translator going between the same languages on different books? How about different translators on the same books to the same other languages?

This might reveal Zipf's law to be an interesting tool for analyzing style and/or language, or it might show that it is not sufficient to get much insight. If the latter, then the next step might be to research what other techniques are used in digital humanities for this kind of analysis.

Possible references:

- Courtney Lawton's dissertation: https://digitalcommons.unl.edu/dissertations/AAI10831194/ (we need to find the full book)
- Isabella Trejo's project: https://github.com/izzytrejo/Zipf

39.7.4 Analyzing wordle

The popular game wordle has spawned many variants, as well as applications to other languages.

In analogy to "opening theory" in chess, it would be interesting to develop a repertoire of best "starting words".

We can try to use shell commands (like grep with regular expressions) and the `/usr/share/dict/words` text file to learn as much as possible about optimal first word choices. We can switch to a full python program once we hit the limits of this approach.

The candidate best words should then be tested on a large monte-carlo sampling of possible challenge words to see which works better.

This can then be expanded in interesting ways to (a) apply to quordle (4 words simultaneously) and larger variants, or (b) to use psychological factors on guessing words to fine-tune the opening words.

To give a bit of a start to see how easy it is to do a command-line study of wordle:

```
# what are all the 5-letter entries in the dictionary?
grep '^.....$' /usr/share/dict/words
# how about if you exclude names and punctuation?
egrep '^.....$' /usr/share/dict/words | egrep -v '[^a-z]'
# many entries are there?
egrep '^.....$' /usr/share/dict/words | egrep -v '[^a-z]' | wc
# what words have 'y' in the 3rd entry?
egrep '^.....$' /usr/share/dict/words | egrep -v '[^a-z]' | grep '..y..'
# how many words have 'y' in the 3rd entry?
egrep '^.....$' /usr/share/dict/words | egrep -v '[^a-z]' | grep '..y..' | wc
# how many words start with 'f' and have 'y' in the 3rd entry?
egrep '^.....$' /usr/share/dict/words | egrep -v '[^a-z]' | grep
'f.y..' | wc
```

Possible references for word lists:

- look at the file /usr/share/dict/words
- https://runestone.academy/ns/books/published/fopp/Projects/common_words.html#common-words

Possible references on other such games:

- https://www.nytimes.com/games/wordle/index.html
- https://worldle.teuteuf.fr/
- https://worldledaily.com/
- https://www.quordle.com/#/
- https://world3dmap.com/duotrigordle-game/
- https://citydle.com/
- https://globle-game.com/
- https://globle-game.com/
- https://greggblanchard.com/statle/
- https://plurality.fun/
- https://googlemapsmania.blogspot.com/2022/06/the-top-10-wordle-like-map-games.html
- https://www.online-tech-tips.com/cool-websites/11-best-sites-to-play-geography-games-online/
- https://locatle.strct.net/

- https://statle.us/
- https://outflux.net/statele/
- https://www.reddit.com/r/geoguessr/comments/v5ihla/locatle_is_a_mix_between_geoguessr_and_wordle/
- explordle
- https://oec.world/en/tradle/

39.7.5 Can generative AI make art or music with a simple project?

Topics: music, visual art

Investigate the various AI engines for which people have written generative AI modules (I think there are some for both tensorflow and pytorch). Only look at free/open-source modules.

Explore the results of generative art and music for those modules, and then see if it is possible to give a simple procedure for someone to start from scratch and end up with running programs that do the work.

CHAPTER

FORTY

APPENDIX: PROPOSED CHAPTERS

Motivation

This appendix is a grabbag of possible chapters – ideas for topics that are interesting, especially to middle and high school students, and which are waiting for someone to write a chapter about them.

It is structured as a simple list of sections with a chapter topic in each. Each section has some references on materials.

Once a co-author wants to write a chapter on that subject, she should transfer that material into a chapter, flesh out the chapter (see Section **??**), and remove it from this appendix.

40.1 A tour of datasets

[status: unwritten]

- https://www.dataquest.io/blog/free-datasets-for-projects/

40.1.1 Climate change

- NOAA weather stations

40.1.2 538 blog data

the 538 github area

40.2 Molecules in three dimensions

First install paraview, that very ambitious volume rendering tool. For molecules we can also install avogadro, jmol, and gdis. And others:

```
sudo apt install paraview avogadro jmol gdis
sudo apt install garlic dssp garlic qutemol viewmol pymol openbabel-gui rasmol
```

The ones that seem to work are paraview, avogadro, jmol, rasmol: you can just run them on a .pdb molecular description file.

You can follow this tutorial for paraview:

https://www.paraview.org/Wiki/The_ParaView_Tutorial

The others probably also have tutorials, but they are less immediately daunting than paraview.

40.2.1 Small Molecules

Start with small molecules, for example from here:

https://ww2.chemistry.gatech.edu/~lw26/structure/small_molecules/index.html

But also at http://www.rcsb.org/pdb/ligand/chemAdvSearch.do you can do a search for smaller molecules. For example I found water with a "search for ligands" in the rcsb.org web site. That gives me:

http://www.rcsb.org/pdb/results/results.do?tabtoshow=Ligand&qrid=9C66910A

which gives, down below, the H2O which is "water" rather than "deuterated water". This lets you download it as:

```
wget http://files.rcsb.org/ligands/view/HOH.cif
```

From that I guessed that I could also get things like Methane and Sulphur atom:

```
wget http://files.rcsb.org/ligands/view/CH4.cif
wget http://files.rcsb.org/ligands/view/S.cif
```

You can also browse like this: go to http://files.rcsb.org/ligands/ and start navigating, for example into C and then C2H. From there you can find:

```
wget http://files.rcsb.org/ligands/view/C2H_ideal.pdb
```

Here are a few more:

```
wget http://files.rcsb.org/ligands/C/C6H/C6H_ideal.pdb
wget http://files.rcsb.org/ligands/C/CH4/CH4_ideal.pdb
```

Then in the search menu you find "Chemical components" and in there "Chemical name"

Now it looks like the way to find molecules by colloquial names is to go to http://ligand-expo.rcsb.org/ld-search.html and do a "molecular name" search, with "molecular name (exact)".

Try putting "Water" and you get this result:

http://ligand-expo.rcsb.org/pyapps/ldHandler.py?formid=cc-index-search&target=Water&operation=name-exact

pick the HOH from there, go to its "Chemical details" and download the "PDB format (ideal coordinates)":

```
wget http://ligand-expo.rcsb.org/reports/H/HOH/HOH_ideal.pdb
jmol HOH_ideal.pdb &
```

and searching for Sucrose (table sugar) you end up at http://ligand-expo.rcsb.org/pyapps/ldHandler.py?formid=cc-index-search&target=Sucrose&operation=name-exact and can download with:

```
wget http://ligand-expo.rcsb.org/reports/S/SUC/SUC_ideal.pdb
jmol SUC_ideal.pdb &
```

40.2.2 Big molecules

Then let's look at big molecules. For example download pdb of this molecule and others at the remarkable rcsb.org site https://www.rcsb.org/structure/6DPA :

```
wget https://files.rcsb.org/download/6DPA.pdb
wget https://files.rcsb.org/download/6YI3.pdb  # seems related to covid-19
```

and load one or all of them in to paraview and the other 3D visualizers, for example with:

```
paraview 6YI3.pdb &
avogadro 6YI3.pdb &
jmol 6YI3.pdb &
rasmol 6YI3.pdb &
```

My first impression: gdis is weird and sometimes goes out to lunch, so I don't understand it enough yet. avogadro and jmol and rasmol are fast and immediate. with paraview you have to do 2 things after loading: (a) enable views by clicking on those two eyes in the left side tree structure, and (b) hit the "apply" button top part of the "properties" tab of the lower left area.

Looking a bit more: avogadro does not work on .cif files, so I'm leaning toward jmol as being the most solid.

40.3 Zeros of polynomials and other functions

Start from our comfortable solution to $ax^2 + bx + c = 0$.

Talk a bit about cubics (how many real/complex roots, …).

Talk a bit about quartics (biquadratic, how many roots, …).

Discuss higher order polynomials and Galois theory.

Introduce numerical methods: Newton's method in particular (connection to calculus).

Extend numerical methods to discuss finding several roots instead of just the closest.

Connect this to optimization, TSP.

40.4 Artificial life

Areas: biology, ecology, complex systems

- https://avida-ed.msu.edu/
- https://en.wikipedia.org/wiki/Avida

40.5 Genetic algorithms

beautiful quote:

> Every boat is copied from another boat… Let's reason as follows in the manner of Darwin. It is clear that a very badly made boat will end up at the bottom after one or two voyages, and thus never be copied… One could then say, with complete rigor, that it is the sea herself who fashions the boats, choosing those which function and destroying the others.
>
> > Propos d'un Normand (1908); as quoted in "Natural selection and cultural rates of change" by D. S. Rogers and P. R. Ehrlich (2008) Proceedings of the National Academy of Sciences 105:3416–3420
> >
> > https://en.wikiquote.org/wiki/%C3%89mile_Chartier
>
> compare that to "coin tossing championship" (from Malcolm Gladwell's "Outliers". discuss polynesian canoes.

Ottawa house of commons: John Sullivan uses evolutionary architecture.

Othello

Robbie the Robot (from Melanie Mitchell's book "Complexity: a Guided Tour")

you'll get to Melanie Mitchell's mooc when you get to chapter 4 on emergent behavior it will point you here: https://www.youtube.com/playlist?list=PLF0b3ThojznRyDQlitfUTzXEXwLNNE-mI and for that section of my book the prerequisite is to watch section 6.3 of the mooc: https://www.youtube.com/watch?v=ZVSseAnEzxs&index=85&list=PLF0b3ThojznRyDQlitfUTzXEXwLNNE-mI watching the full MOOC and the section on Robbie the Robot should be done at home since it goes somewhat afield of what we are doing at work

Other resources:

- GAs on the TSP: https://cs.stanford.edu/people/eroberts/courses/soco/projects/1997-98/genetic-algorithms/apps.html
- Book:
  - Genetic Algorithms in Python
- Examples:

  https://www.codeproject.com/Articles/1104747/Introduction-to-Genetic-Algorithms-with-Python-Hel
- For how to encode a game strategy:
  - To practice Othello: learn here: https://www.eothello.com/ and play here: https://hewgill.com/othello/
  - https://pdfs.semanticscholar.org/9fbb/a0583259b70e318003f5b6861faf9447060f.pdf
  - https://www.codingame.com/blog/genetic-algorithms-coders-strike-back-game/
  - This does a full 9-square-puzzle game encoding: https://www.researchgate.net/publication/220176829_Applying_genetic_algorithms_to_game_search_trees
  - tic-tac-toe: http://www.genetic-programming.org/sp2003/Hochmuth.pdf
  - https://www.cs.auckland.ac.nz/research/gameai/projects/GA%20in%20FreeCiv.pdf
  - othello: https://www.researchgate.net/publication/2599927_EVOLVING_COMPLEX_OTHELLO_STRATEGIES_USING_MARKER-BASED_GENETIC_ENCODING_OF_NEURAL_NETWORKS_David_Moriarty_and_Risto_Miikkulainen
  - programming it in Python and knapsack problem: https://blog.sicara.com/getting-started-genetic-algorithms-python-tutorial-81ffa1dd72f9

  - this goes in to chess; mentions othello in passing, and has an appendix on Elo ratings formulae https://arxiv.org/pdf/1711.08337.pdf
  - this has interesting discussions, but is ultimately about neural networks and thus harder than what I'm thinking of for this chapter. http://nn.cs.utexas.edu/?moriarty:discovering, http://nn.cs.utexas.edu/downloads/papers/moriarty.discovering.pdf – still, Moriarty and Miikkulainen do an interesting job of explaining why humans don't see some of the machine moves.
  - the previous article is one in this site: http://satirist.org/learn-game/methods/ga/marker.html – this includes http://nn.cs.utexas.edu/downloads/papers/fullmer.genetic-encoding.pdf which seems to be about encodings.
  - this looks like a very nice intro to encoding an othello GA: https://pdfs.semanticscholar.org/ca88/76e8d8232403dfa4dcb75c67dd1031dd6bf3.pdf
  - other othello resource: https://github.com/blanyal/alpha-zero
  - much more academic stuff at https://scholar.google.com/scholar?hl=en&as_sdt=0%2C32&as_vis=1&q=othello+genetic+algorithm&btnG=
- Animated games: https://luckytoilet.wordpress.com/2011/05/27/coding-a-tetris-ai-using-a-genetic-algorithm/
- Evolving cellular automata:
  - https://arxiv.org/pdf/adap-org/9303003.pdf

Projects to work on: [… unfinished …]

40.6 Fourier Analysis

This is largely already written in my old latex book. I just need to bring it in here, and revise what I think of the prerequisites.

Latest: this has mostly been moved to the "mark working group" book.

40.7 Simpson's paradox

https://qr.ae/TWNhZd?share=1 https://www.quora.com/What-is-Simpsons-paradox?share=1

CHAPTER

FORTYONE

COPYING AND LEGAL MATTERS

41.1 Copyright

> Copyright (C) 2017-2022 Mark Galassi, Leina Gries, Sophia Mulholland, Almond Heil.

41.2 License for the book

> This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

We reproduce the full text of the CC BY-SA license below.

41.3 License for code samples

All the software code samples are free software, and can be freely redistributed under the GNU General Public License (GPL), version 3.

We reproduce the full text of the GPL below.

41.4 CC BY-SA 4.0 license

```
Attribution-ShareAlike 4.0 International

=======================================================================

Creative Commons Corporation ("Creative Commons") is not a law firm and
does not provide legal services or legal advice. Distribution of
Creative Commons public licenses does not create a lawyer-client or
other relationship. Creative Commons makes its licenses and related
information available on an "as-is" basis. Creative Commons gives no
warranties regarding its licenses, any material licensed under their
terms and conditions, or any related information. Creative Commons
disclaims all liability for damages resulting from their use to the
fullest extent possible.

```

(continues on next page)

(continued from previous page)

```
Using Creative Commons Public Licenses

Creative Commons public licenses provide a standard set of terms and
conditions that creators and other rights holders may use to share
original works of authorship and other material subject to copyright
and certain other rights specified in the public license below. The
following considerations are for informational purposes only, are not
exhaustive, and do not form part of our licenses.

     Considerations for licensors: Our public licenses are
     intended for use by those authorized to give the public
     permission to use material in ways otherwise restricted by
     copyright and certain other rights. Our licenses are
     irrevocable. Licensors should read and understand the terms
     and conditions of the license they choose before applying it.
     Licensors should also secure all rights necessary before
     applying our licenses so that the public can reuse the
     material as expected. Licensors should clearly mark any
     material not subject to the license. This includes other CC-
     licensed material, or material used under an exception or
     limitation to copyright. More considerations for licensors:
    wiki.creativecommons.org/Considerations_for_licensors

     Considerations for the public: By using one of our public
     licenses, a licensor grants the public permission to use the
     licensed material under specified terms and conditions. If
     the licensor's permission is not necessary for any reason--for
     example, because of any applicable exception or limitation to
     copyright--then that use is not regulated by the license. Our
     licenses grant only permissions under copyright and certain
     other rights that a licensor has authority to grant. Use of
     the licensed material may still be restricted for other
     reasons, including because others have copyright or other
     rights in the material. A licensor may make special requests,
     such as asking that all changes be marked or described.
     Although not required by our licenses, you are encouraged to
     respect those requests where reasonable. More considerations
     for the public:
    wiki.creativecommons.org/Considerations_for_licensees

=======================================================================

Creative Commons Attribution-ShareAlike 4.0 International Public
License

By exercising the Licensed Rights (defined below), You accept and agree
to be bound by the terms and conditions of this Creative Commons
Attribution-ShareAlike 4.0 International Public License ("Public
License"). To the extent this Public License may be interpreted as a
contract, You are granted the Licensed Rights in consideration of Your
acceptance of these terms and conditions, and the Licensor grants You
such rights in consideration of benefits the Licensor receives from
```

(continues on next page)

(continued from previous page)

```
making the Licensed Material available under these terms and
conditions.


Section 1 -- Definitions.

  a. Adapted Material means material subject to Copyright and Similar
     Rights that is derived from or based upon the Licensed Material
     and in which the Licensed Material is translated, altered,
     arranged, transformed, or otherwise modified in a manner requiring
     permission under the Copyright and Similar Rights held by the
     Licensor. For purposes of this Public License, where the Licensed
     Material is a musical work, performance, or sound recording,
     Adapted Material is always produced where the Licensed Material is
     synched in timed relation with a moving image.

  b. Adapter's License means the license You apply to Your Copyright
     and Similar Rights in Your contributions to Adapted Material in
     accordance with the terms and conditions of this Public License.

  c. BY-SA Compatible License means a license listed at
     creativecommons.org/compatiblelicenses, approved by Creative
     Commons as essentially the equivalent of this Public License.

  d. Copyright and Similar Rights means copyright and/or similar rights
     closely related to copyright including, without limitation,
     performance, broadcast, sound recording, and Sui Generis Database
     Rights, without regard to how the rights are labeled or
     categorized. For purposes of this Public License, the rights
     specified in Section 2(b)(1)-(2) are not Copyright and Similar
     Rights.

  e. Effective Technological Measures means those measures that, in the
     absence of proper authority, may not be circumvented under laws
     fulfilling obligations under Article 11 of the WIPO Copyright
     Treaty adopted on December 20, 1996, and/or similar international
     agreements.

  f. Exceptions and Limitations means fair use, fair dealing, and/or
     any other exception or limitation to Copyright and Similar Rights
     that applies to Your use of the Licensed Material.

  g. License Elements means the license attributes listed in the name
     of a Creative Commons Public License. The License Elements of this
     Public License are Attribution and ShareAlike.

  h. Licensed Material means the artistic or literary work, database,
     or other material to which the Licensor applied this Public
     License.

  i. Licensed Rights means the rights granted to You subject to the
     terms and conditions of this Public License, which are limited to
```

(continues on next page)

(continued from previous page)

```
     all Copyright and Similar Rights that apply to Your use of the
     Licensed Material and that the Licensor has authority to license.

  j. Licensor means the individual(s) or entity(ies) granting rights
     under this Public License.

  k. Share means to provide material to the public by any means or
     process that requires permission under the Licensed Rights, such
     as reproduction, public display, public performance, distribution,
     dissemination, communication, or importation, and to make material
     available to the public including in ways that members of the
     public may access the material from a place and at a time
     individually chosen by them.

  l. Sui Generis Database Rights means rights other than copyright
     resulting from Directive 96/9/EC of the European Parliament and of
     the Council of 11 March 1996 on the legal protection of databases,
     as amended and/or succeeded, as well as other essentially
     equivalent rights anywhere in the world.

  m. You means the individual or entity exercising the Licensed Rights
     under this Public License. Your has a corresponding meaning.


Section 2 -- Scope.

  a. License grant.

       1. Subject to the terms and conditions of this Public License,
          the Licensor hereby grants You a worldwide, royalty-free,
          non-sublicensable, non-exclusive, irrevocable license to
          exercise the Licensed Rights in the Licensed Material to:

            a. reproduce and Share the Licensed Material, in whole or
               in part; and

            b. produce, reproduce, and Share Adapted Material.

       2. Exceptions and Limitations. For the avoidance of doubt, where
          Exceptions and Limitations apply to Your use, this Public
          License does not apply, and You do not need to comply with
          its terms and conditions.

       3. Term. The term of this Public License is specified in Section
          6(a).

       4. Media and formats; technical modifications allowed. The
          Licensor authorizes You to exercise the Licensed Rights in
          all media and formats whether now known or hereafter created,
          and to make technical modifications necessary to do so. The
          Licensor waives and/or agrees not to assert any right or
          authority to forbid You from making technical modifications
```

(continues on next page)

(continued from previous page)

```
          necessary to exercise the Licensed Rights, including
          technical modifications necessary to circumvent Effective
          Technological Measures. For purposes of this Public License,
          simply making modifications authorized by this Section 2(a)
          (4) never produces Adapted Material.

       5. Downstream recipients.

            a. Offer from the Licensor -- Licensed Material. Every
               recipient of the Licensed Material automatically
               receives an offer from the Licensor to exercise the
               Licensed Rights under the terms and conditions of this
               Public License.

            b. Additional offer from the Licensor -- Adapted Material.
               Every recipient of Adapted Material from You
               automatically receives an offer from the Licensor to
               exercise the Licensed Rights in the Adapted Material
               under the conditions of the Adapter's License You apply.

            c. No downstream restrictions. You may not offer or impose
               any additional or different terms or conditions on, or
               apply any Effective Technological Measures to, the
               Licensed Material if doing so restricts exercise of the
               Licensed Rights by any recipient of the Licensed
               Material.

       6. No endorsement. Nothing in this Public License constitutes or
          may be construed as permission to assert or imply that You
          are, or that Your use of the Licensed Material is, connected
          with, or sponsored, endorsed, or granted official status by,
          the Licensor or others designated to receive attribution as
          provided in Section 3(a)(1)(A)(i).

  b. Other rights.

       1. Moral rights, such as the right of integrity, are not
          licensed under this Public License, nor are publicity,
          privacy, and/or other similar personality rights; however, to
          the extent possible, the Licensor waives and/or agrees not to
          assert any such rights held by the Licensor to the limited
          extent necessary to allow You to exercise the Licensed
          Rights, but not otherwise.

       2. Patent and trademark rights are not licensed under this
          Public License.

       3. To the extent possible, the Licensor waives any right to
          collect royalties from You for the exercise of the Licensed
          Rights, whether directly or through a collecting society
          under any voluntary or waivable statutory or compulsory
          licensing scheme. In all other cases the Licensor expressly
```

(continues on next page)

(continued from previous page)

```
          reserves any right to collect such royalties.


Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the
following conditions.

  a. Attribution.

       1. If You Share the Licensed Material (including in modified
          form), You must:

            a. retain the following if it is supplied by the Licensor
               with the Licensed Material:

                 i. identification of the creator(s) of the Licensed
                    Material and any others designated to receive
                    attribution, in any reasonable manner requested by
                    the Licensor (including by pseudonym if
                    designated);

                ii. a copyright notice;

               iii. a notice that refers to this Public License;

                iv. a notice that refers to the disclaimer of
                    warranties;

                 v. a URI or hyperlink to the Licensed Material to the
                    extent reasonably practicable;

            b. indicate if You modified the Licensed Material and
               retain an indication of any previous modifications; and

            c. indicate the Licensed Material is licensed under this
               Public License, and include the text of, or the URI or
               hyperlink to, this Public License.

       2. You may satisfy the conditions in Section 3(a)(1) in any
          reasonable manner based on the medium, means, and context in
          which You Share the Licensed Material. For example, it may be
          reasonable to satisfy the conditions by providing a URI or
          hyperlink to a resource that includes the required
          information.

       3. If requested by the Licensor, You must remove any of the
          information required by Section 3(a)(1)(A) to the extent
          reasonably practicable.

  b. ShareAlike.
```

(continues on next page)

(continued from previous page)

```
     In addition to the conditions in Section 3(a), if You Share
     Adapted Material You produce, the following conditions also apply.

       1. The Adapter's License You apply must be a Creative Commons
          license with the same License Elements, this version or
          later, or a BY-SA Compatible License.

       2. You must include the text of, or the URI or hyperlink to, the
          Adapter's License You apply. You may satisfy this condition
          in any reasonable manner based on the medium, means, and
          context in which You Share Adapted Material.

       3. You may not offer or impose any additional or different terms
          or conditions on, or apply any Effective Technological
          Measures to, Adapted Material that restrict exercise of the
          rights granted under the Adapter's License You apply.


Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that
apply to Your use of the Licensed Material:

  a. for the avoidance of doubt, Section 2(a)(1) grants You the right
     to extract, reuse, reproduce, and Share all or a substantial
     portion of the contents of the database;

  b. if You include all or a substantial portion of the database
     contents in a database in which You have Sui Generis Database
     Rights, then the database in which You have Sui Generis Database
     Rights (but not its individual contents) is Adapted Material,

     including for purposes of Section 3(b); and
  c. You must comply with the conditions in Section 3(a) if You Share
     all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not
replace Your obligations under this Public License where the Licensed
Rights include other Copyright and Similar Rights.


Section 5 -- Disclaimer of Warranties and Limitation of Liability.

  a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE
     EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS
     AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF
     ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS,
     IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION,
     WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR
     PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS,
     ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT
     KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT
```

(continues on next page)

(continued from previous page)

```
     ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

  b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE
     TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION,
     NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT,
     INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES,
     COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR
     USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN
     ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR
     DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR
     IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

  c. The disclaimer of warranties and limitation of liability provided
     above shall be interpreted in a manner that, to the extent
     possible, most closely approximates an absolute disclaimer and
     waiver of all liability.


Section 6 -- Term and Termination.

  a. This Public License applies for the term of the Copyright and
     Similar Rights licensed here. However, if You fail to comply with
     this Public License, then Your rights under this Public License
     terminate automatically.

  b. Where Your right to use the Licensed Material has terminated under
     Section 6(a), it reinstates:

       1. automatically as of the date the violation is cured, provided
          it is cured within 30 days of Your discovery of the
          violation; or

       2. upon express reinstatement by the Licensor.

     For the avoidance of doubt, this Section 6(b) does not affect any
     right the Licensor may have to seek remedies for Your violations
     of this Public License.

  c. For the avoidance of doubt, the Licensor may also offer the
     Licensed Material under separate terms or conditions or stop
     distributing the Licensed Material at any time; however, doing so
     will not terminate this Public License.

  d. Sections 1, 5, 6, 7, and 8 survive termination of this Public
     License.


Section 7 -- Other Terms and Conditions.

  a. The Licensor shall not be bound by any additional or different
     terms or conditions communicated by You unless expressly agreed.

```

(continues on next page)

(continued from previous page)

```
  b. Any arrangements, understandings, or agreements regarding the
     Licensed Material not stated herein are separate from and
     independent of the terms and conditions of this Public License.


Section 8 -- Interpretation.

  a. For the avoidance of doubt, this Public License does not, and
     shall not be interpreted to, reduce, limit, restrict, or impose
     conditions on any use of the Licensed Material that could lawfully
     be made without permission under this Public License.

  b. To the extent possible, if any provision of this Public License is
     deemed unenforceable, it shall be automatically reformed to the
     minimum extent necessary to make it enforceable. If the provision
     cannot be reformed, it shall be severed from this Public License
     without affecting the enforceability of the remaining terms and
     conditions.

  c. No term or condition of this Public License will be waived and no
     failure to comply consented to unless expressly agreed to by the
     Licensor.

  d. Nothing in this Public License constitutes or may be interpreted
     as a limitation upon, or waiver of, any privileges and immunities
     that apply to the Licensor or You, including from the legal
     processes of any jurisdiction or authority.


=======================================================================

Creative Commons is not a party to its public
licenses. Notwithstanding, Creative Commons may elect to apply one of
its public licenses to material it publishes and in those instances
will be considered the “Licensor.” The text of the Creative Commons
public licenses is dedicated to the public domain under the CC0 Public
Domain Dedication. Except for the limited purpose of indicating that
material is shared under a Creative Commons public license or as
otherwise permitted by the Creative Commons policies published at
creativecommons.org/policies, Creative Commons does not authorize the
use of the trademark "Creative Commons" or any other trademark or logo
of Creative Commons without its prior written consent including,
without limitation, in connection with any unauthorized modifications
to any of its public licenses or any other arrangements,
understandings, or agreements concerning use of licensed material. For
the avoidance of doubt, this paragraph does not form part of the
public licenses.

Creative Commons may be contacted at creativecommons.org.
```

41.5 GNU General Public License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
```

(continues on next page)

(continued from previous page)

```
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.
```

(continues on next page)

(continued from previous page)

```
  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.
```

(continues on next page)

(continued from previous page)

```
  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.
```

(continues on next page)

(continued from previous page)

```
  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
```

(continues on next page)

(continued from previous page)

```
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
```

(continues on next page)

(continued from previous page)

```
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
```

(continues on next page)

(continued from previous page)

```
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).
```

(continues on next page)

(continued from previous page)

```
  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
```

(continues on next page)

(continued from previous page)

```
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
```

(continues on next page)

(continued from previous page)

```
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
```

(continues on next page)

(continued from previous page)

```
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS
```

(continues on next page)

(continued from previous page)

```
            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<https://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<https://www.gnu.org/licenses/why-not-lgpl.html>.
```

CHAPTER FORTYTWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[Bac77] J Backus. Musical note to frequency conversion chart. http://www.audiology.org/sites/default/files/ChasinConversionChart.pdf, 1977. Accessed: 2016-05-06.

[Bir15] Alistair Bird. Apiological: mathematical speculations about bees (part 3: travelling salesman). http://aperiodical.com/2015/03/apiological-part-3/, 2015. Accessed: 2015-03-19.

[Gal15] Mark Galassi. *Hacking Camp Teacher's Manual*. web, 2015.

[Gal16] Mark Galassi. *Scientific Computing for Kids*. web, 2016.

[Gra12] Christopher M Graney. Doubting, testing, and confirming galileo: a translation of giovanni battista riccioli's experiments regarding the motion of a falling body, as reported in his 1651 almagestum novum. *arXiv preprint arXiv:1204.3267*, 2012.

[Pin11] Steven Pinker. *The better angels of our nature: Why violence has declined*. Penguin, 2011.

[RR10] Carmen M Reinhart and Kenneth S Rogoff. Growth in a time of debt (digest summary). *American Economic Review*, 100(2):573–578, 2010.

[ZJZ+16] Caiping Zhang, Jiuchun Jiang, Linjing Zhang, Sijia Liu, Leyi Wang, and Poh Chiang Loh. A generalized soc-ocv model for lithium-ion batteries and the soc estimation for lnmco battery. *Energies*, 9(11):900, 2016.