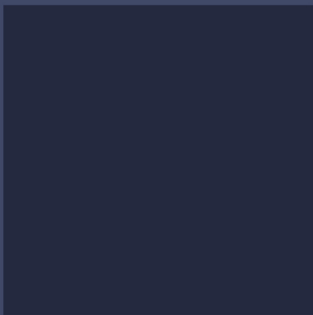


# Science and Computing with Raspberry Pi

**Brian R Kent**



# Science and Computing with Raspberry Pi



# Science and Computing with Raspberry Pi

**Brian R Kent, PhD**

*National Radio Astronomy Observatory, Charlottesville, Virginia, USA*

Morgan & Claypool Publishers



Copyright © 2018 Morgan & Claypool Publishers

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher, or as expressly permitted by law or under terms agreed with the appropriate rights organization. Multiple copying is permitted in accordance with the terms of licences issued by the Copyright Licensing Agency, the Copyright Clearance Centre and other reproduction rights organisations.

#### Rights & Permissions

To obtain permission to re-use copyrighted material from Morgan & Claypool Publishers, please contact [info@morganclaypool.com](mailto:info@morganclaypool.com).

ISBN 978-1-6817-4996-9 (ebook)

ISBN 978-1-6817-4993-8 (print)

ISBN 978-1-6817-4994-5 (mobi)

DOI 10.1088/978-1-6817-4996-9

Version: 20180601

IOP Concise Physics

ISSN 2053-2571 (online)

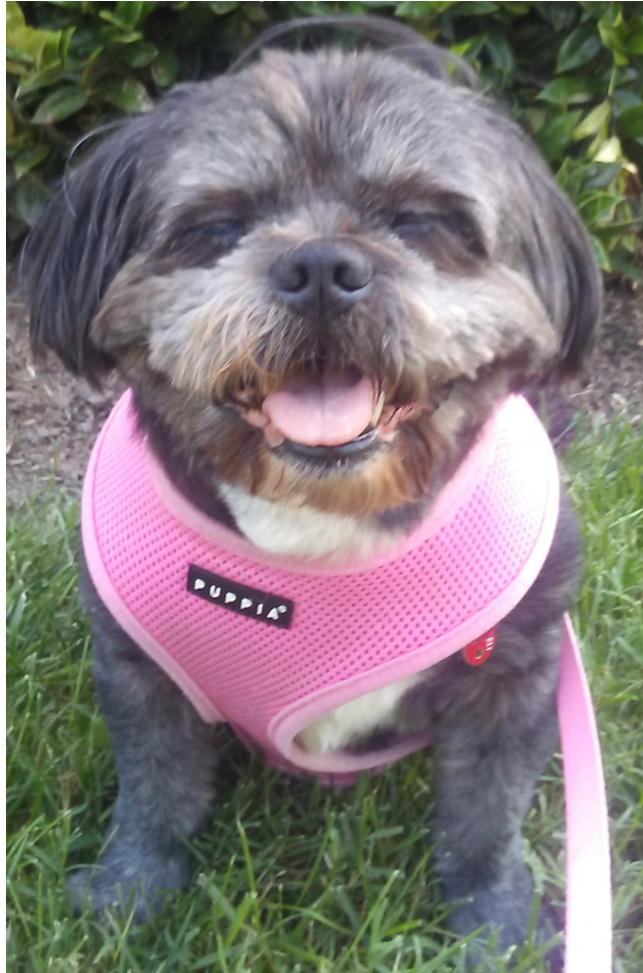
ISSN 2054-7307 (print)

A Morgan & Claypool publication as part of IOP Concise Physics

Published by Morgan & Claypool Publishers, 1210 Fifth Avenue, Suite 250, San Rafael, CA, 94901, USA

IOP Publishing, Temple Circus, Temple Way, Bristol BS1 6HG, UK

*This book is dedicated to Snickers.*





# Contents

<b>Preface</b>	<b>ix</b>
<b>Acknowledgements</b>	<b>x</b>
<b>Author biography</b>	<b>xi</b>
<b>1 Raspberry Pi</b>	<b>1-1</b>
1.1 Single-board computing	1-1
1.2 Why Raspberry Pi?	1-2
Reference	1-3
<b>2 Setting up your system</b>	<b>2-1</b>
2.1 Hardware configuration, requirements, and limitations	2-1
2.2 Understanding Linux	2-1
2.3 Python	2-2
2.4 <i>Mathematica</i> and Wolfram Alpha	2-5
2.5 Sources of astronomical science data	2-9
2.6 Using revision control	2-9
2.7 Jupyter notebooks	2-10
2.8 Coding pedagogy	2-11
References	2-13
<b>3 Chaos and non-linear dynamics</b>	<b>3-1</b>
3.1 One and two dimensional pseudo random walks	3-1
3.2 Logistic maps, bifurcation, and chaos	3-6
3.3 Cellular automata	3-12
References	3-17
<b>4 Physics and astronomy</b>	<b>4-1</b>
4.1 A simple pendulum	4-1
4.2 The double pendulum	4-2
4.3 Hydrostatics	4-6
4.4 Astronomical catalogs	4-8
4.5 The Lane–Emden equation	4-12
4.6 Radiative transfer	4-14
References	4-16

<b>5</b>	<b>Machine learning</b>	<b>5-1</b>
5.1	Spanning trees	5-1
5.2	Neural networks and classification	5-7
	References	5-10
<b>6</b>	<b>Image combination and analysis</b>	<b>6-1</b>
6.1	Image manipulation	6-1
6.2	Creating a multi-wavelength astronomical image	6-3
6.3	Manipulating astronomical data cubes	6-6
	References	6-10
<b>Appendices</b>		
<b>A</b>	<b><i>Mathematica</i> shortcuts and help</b>	<b>A-1</b>
<b>B</b>	<b>Important Python modules and resources</b>	<b>B-1</b>

# Preface

This is a book written for the motivated individual who wishes to see what kinds of computational experiments they can explore with the Raspberry Pi single-board computer. The reader will learn about two languages that are accessible on the Raspberry Pi—*Python* and *Mathematica*<sup>®</sup>. Both have their own extensive forum of users and are widely used by the scientific and technical communities to solve problems in science every day. This book is not an exhaustive resource for either language nor is the Raspberry Pi the only hardware platform where the examples can be explored. However it represents a unique entry into scientific computing accessible to everyone. Through simple, introductory exercises in physics, astronomy, chaos theory, and machine learning, this text aims to be a launching point for the beginner into a wide range of exciting computing paradigms.

# Acknowledgements

Raspberry Pi is a trademark of the Raspberry Pi Foundation. *Mathematica*<sup>®</sup> is a registered trademark of Wolfram Research, Inc. Bitbucket<sup>®</sup> is a registered trademark of Atlassian. The author would like to thank the IOP Concise Physics series team of Jeanine Burke, Melanie Carlson, and Joel Claypool of Morgan & Claypool Publishers. The data cube section is the result of discussions with Dr Jeff Mangum of the NRAO. We acknowledge the use of NASA's SkyView facility (<http://skyview.gsfc.nasa.gov>) located at NASA Goddard Space Flight Center. The National Radio Astronomy Observatory (NRAO) is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

# Author biography

## Brian R Kent

---



Brian R Kent, PhD, is a scientist with the National Radio Astronomy Observatory in Charlottesville, Virginia. His publications and studies in astrophysics and computing include scientific visualizations of a variety of theoretical and observational phenomena. He is interested in visualizing data for scientific analysis, 3D graphics, and introducing scientific programming via single-board computers like Raspberry Pi. Dr Kent received

his PhD in Astronomy and Space Sciences from Cornell University. His website is <http://www.cv.nrao.edu/~bkent/>.



# Science and Computing with Raspberry Pi

Brian R Kent

---

## Chapter 1

### Raspberry Pi

#### 1.1 Single-board computing

Single-board computing gives researchers, hobbyists, and educators a programming platform that is agile for rapid development and prototyping. Students can develop code in a variety of environments and languages. Scientific software and computing now relies on using sound software engineering principles to further the field and process data from experiments and simulations. While single-board computers are not geared toward big data processing, they certainly can be utilized as versatile test-bed environments. Such systems have the ability to introduce users to new development environments while being cost effective at the same time. Their utility to cost ratio is very high.

The variety of applications for single-board computers stretches from hardware and camera control to building proof-of-concept clusters. Science, technology, engineering, and mathematics (STEM) education initiatives in particular can benefit from the low barrier to entry in obtaining and setting up a single-board computer.

Single-board computers can be used as a standalone hardware solution or integrated into larger systems. Applications for single-board computers include media center solutions, embedded as central controllers for unmanned aerial systems, and in many areas of robotics—anywhere a small form factor computer is needed. As a computing platform they can be used as an outstanding introduction to programming and software engineering for education.

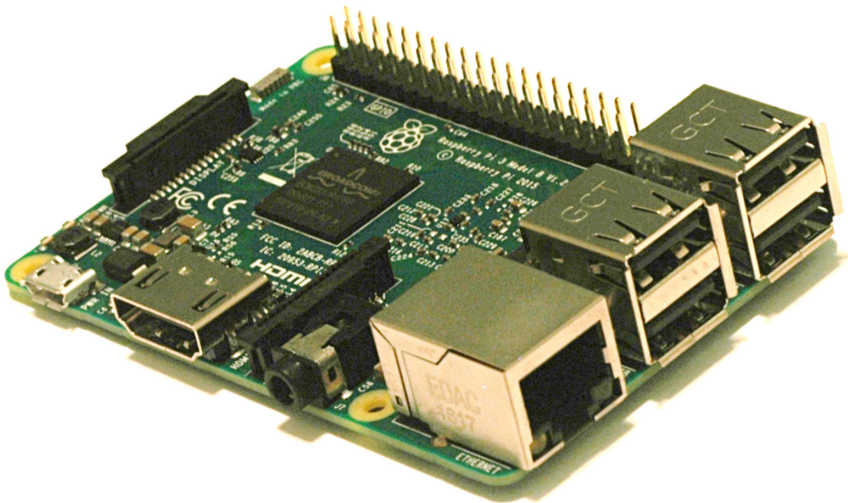
While there are many books, videos, and tutorials on using single-board computers for a variety of hardware and peripheral projects, the focus of this Concise Physics series book is to introduce readers to using the Raspberry Pi as a simple development platform for scientific computing. The book will focus on understanding the capabilities of the Raspberry Pi using both Python and *Mathematica*<sup>®</sup> environments in the native Raspbian Linux operating system.

## 1.2 Why Raspberry Pi?

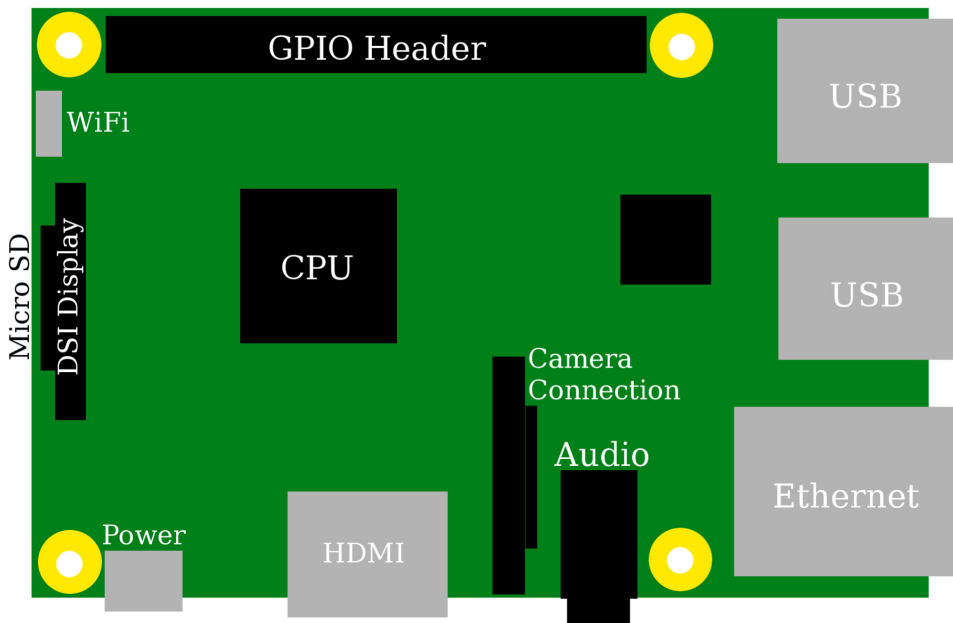
The Raspberry Pi (or RPi) single-board computer was conceived as a low-cost hardware solution to teach basic computing and software programming. The RPi started with the Model A in 2013 with a 700 MHz single-core processor. The suite of RPis has grown to include the B and B+ models with 1.2 GHz and 1.4 GHz 64-bit quad-core processors respectively as well as the entry level RPi Zero, both of which include wireless network interfaces. The third-generation Model B is shown in figure 1.1.

The architecture of the RPi 3 Model B and B+ is shown in the figure 1.2 schematic. The full featured layout has four USB ports for peripherals, HDMI display output and audio, as well as support for cameras and other hardware interfaces. While the RPi can be purchased as a standalone board, most kits include a microSD card with NOOBS and Raspbian, a protective case, an HDMI cable, and a power cord. The other three items the reader will need to acquire are a keyboard, mouse, and display with an HDMI port.

As with many single-board computer solutions, the RPi has been successfully implemented in hardware projects ranging from controlling robots to weather monitoring stations to automating processes of your home environment in the ‘internet of things’ age. Innovators are even building small RPi compute clusters [1]. The original purpose was to provide a low cost system so that students could learn programming. Its rapid adoption and broad community make it an excellent platform for students to understand the basics of scientific programming and have fun learning by example. What does the future hold for STEM education and computing as we can fit more capabilities onto an RPi?



**Figure 1.1.** Image of the Raspberry Pi 3 Model B single-board computer. This small board has the capability of being the reader’s entry into scientific computing. All exercises presented in this book are capable of being run on this seemingly small bit of computer hardware.



**Figure 1.2.** Diagram of the Raspberry Pi 3 Model B and B+ single-board computer. The hardware layout schematic here shows all the features that will be needed. While we will not be using the camera connection or GPIO Header, these are labeled here as they allow a Raspberry Pi user to attach additional peripheral elements to the board.

## Reference

- [1] Mappuji A, Effendy N, Mustaghfirin M, Sondok F, Yuniar R P and Pangesti S P 2016 Study of raspberry pi 2 quad-core cortex-a7 cpu cluster as a mini supercomputer In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pp 1-4

# Science and Computing with Raspberry Pi

Brian R Kent

---

## Chapter 2

### Setting up your system

#### 2.1 Hardware configuration, requirements, and limitations

The Raspberry Pi 3 module requires keyboard and mouse peripherals for user input. A microSD card with NOOBS (New Out Of Box Software) and the Raspbian operating system are a user's most efficient pathway to getting started. In addition, an HDMI connection and display monitor or panel are needed. All exercises in this book are well within the computational and memory usage specifications of the Raspberry Pi.

Upon inserting the NOOBS microSD card and providing power to the RPi for the first time, you will be prompted to install the operating system. The Raspbian desktop will look similar to figure 2.1. It can easily drive a 1080p high-definition screen.

It is recommended for your system's security to change the default system 'raspberry' password for the 'pi' user account, or better yet, to create a separate user account with a secure password. This can be done with the command **passwd** in a terminal window. It is also important to update any installed packages via<sup>1</sup>

```
sudo apt-get update
sudo apt-get dist-upgrade
```

#### 2.2 Understanding Linux

The Raspbian operating system<sup>2</sup> is based on Debian<sup>3</sup> built on the Linux kernel. The Linux kernel is the basis for more computer desktop, laptop, server, and mobile systems than any other software technology in the world<sup>4</sup>. By clicking on the

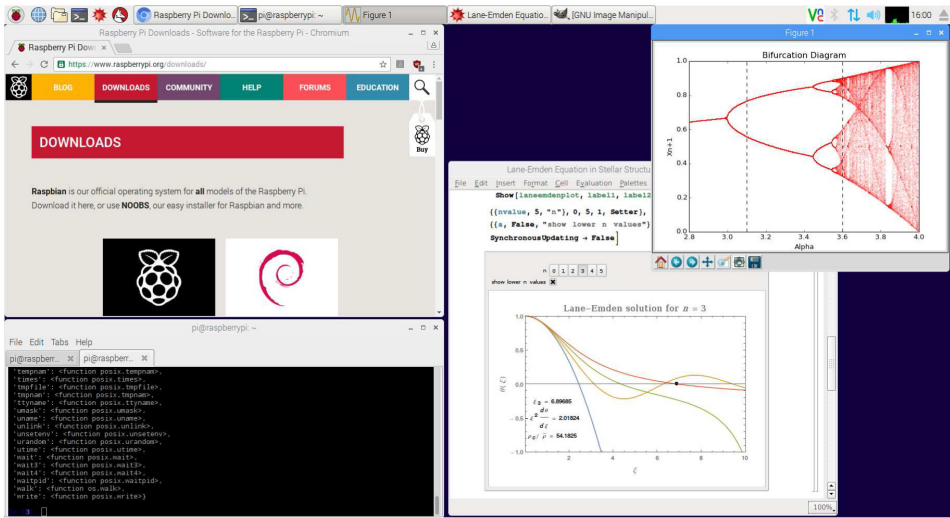
---

<sup>1</sup> <https://www.raspberrypi.org/documentation/raspbian/updating.md>

<sup>2</sup> <https://www.raspberrypi.org/downloads/raspbian/>

<sup>3</sup> <https://www.debian.org/>

<sup>4</sup> <http://statcounter.com/>; <https://www.linuxcounter.net/>



**Figure 2.1.** The Raspbian desktop and Linux terminal window. Standard icons in the upper left are customizable, but include selections for turning the device on and off, opening a terminal, updating the Raspbian OS, and starting *Mathematica*.

terminal icon in the upper left, one can open a Linux terminal window. Most interactions will occur through the *command line interface (cli)*. The following basic Linux commands will serve the user well in their new Raspberry Pi explorations.

---

<b>ls</b>	-list the contents of a directory
<b>cd</b>	-change to a named directory [cd <directory name>]
<b>pwd</b>	-return the present working directory
<b>man</b>	-manual pages for any Linux commands
<b>mkdir</b>	-make a directory
<b>rmdir</b>	-remove an empty directory
<b>rm</b>	-remove a file
<b>mv</b>	-move or rename a file
<b>clear</b>	-clear the terminal window, return prompt to top
<b>find/locate</b>	-search commands
<b>history</b>	-review the terminal history of commands one has executed.

---

## 2.3 Python

Python is a high-level language released in 1991 that provides users with many paradigms of programming and easily readable code<sup>5</sup>. Code is easily obtained and

<sup>5</sup><http://www.python.org/>

downloaded with the Python Package Index<sup>6</sup>. The wide variety of well-maintained and robust technical modules make the Python style of programming advantageous to many areas of scientific computing. The language lends itself well to structured object-oriented programming for maintainable code as well as simple scripts for data reduction.

A number of Python modules are very useful for data analysis, scientific graphics and plotting, and as development environments. While not all of these will be used in this book, it is worthwhile for the reader to explore these options depending on their computing and analysis needs.

---

<b>IPython</b>	-interactive command line package with support for notebooks <a href="https://ipython.org/">https://ipython.org/</a>
<b>Matplotlib</b>	-a graphics and plotting library <a href="https://matplotlib.org/">https://matplotlib.org/</a>
<b>pandas</b>	-especially useful for time series manipulation <a href="https://pandas.pydata.org/">https://pandas.pydata.org/</a>
<b>Scikit-learn</b>	-machine learning library <a href="https://scikit-learn.org">https://scikit-learn.org</a>
<b>SciPy</b>	-scientific Python and computing, line fitting, ODE solvers, etc <a href="https://www.scipy.org/">https://www.scipy.org/</a>
<b>PyCharm</b>	-integrated development environment for Python <a href="https://www.jetbrains.com/pycharm/">https://www.jetbrains.com/pycharm/</a>
<b>Blender</b>	-3D graphics, visualization, and animation [1] <a href="https://www.blender.org/">https://www.blender.org/</a>
<b>AstroPy</b>	-astronomy data manipulation library <a href="http://www.astropy.org/">http://www.astropy.org/</a>
<b>Anaconda</b>	-an all-in-one Python distribution <a href="https://anaconda.org/">https://anaconda.org/</a>

---

Python modules can be added via the Debian package installer<sup>7</sup>:

```
sudo apt-get install python-numpy python-scipy python-matplotlib
    ipython ipython-notebook python-pandas python-sympy python-nose
```

A basic Python prompt can be started by typing **python** in a terminal. However, the IPython prompt (started with '**ipython**') gives the user much more utility with features at the command prompt including tab completion and syntax highlighting.

With creating a new \*.py script or interactively inputting commands on the IPython prompt, modules can be imported with the command 'import' in the file

<sup>6</sup><https://pypi.python.org/pypi>

<sup>7</sup><https://www.scipy.org/install.html>

prelude. The structure of a program written in Python can begin with these commands

```
import os
import math
import sys
import glob
import pdb
```

If one wishes to import a specific class or method from a module

```
from math import log
```

Or use an alias for brevity

```
import matplotlib.pyplot as plt
```

The Python debugger (**pdb**) is a useful module for stepping through and troubleshooting possible code issues. The following debugger command will set a break point in your code at which point one can examine defined variables that are in scope.

```
import pdb
pdb.set_trace()
```

These Python items and others will be used in the scientific computing exercises demonstrated in later chapters.

Python modules adhere to the paradigm of *object oriented programming* (OOP). In fact, everything in Python is an object, and each module contains *classes*. A class is like a blueprint or template for a programming object. An object is an instance of the class type that contains the methods and variables associate with that class [3].

After importing a Python module, we can look at the methods with `<module>.__dict__`

```
import sys
sys.__dict__.keys()
['setrecursionlimit',
'dont_write_bytecode',
'getrefcount',
'long_info',
'path_importer_cache',
'stdout',
...]
```

This gives us a place to start when examining a module for the first time. The aforementioned tab completion will also show available methods within a module.

The code presented in each chapter is meant to be copied and pasted into an IPython session so that the new user can study and think about what each piece is doing. When executing a Python script, one need only input the command at the IPython prompt

```
execfile('script.py')
```

Alternatively, multiple lines of code can be input at the IPython prompt with the command **cpaste**, followed by two dashes (- -). The following code shows the components of a simple Python script and damped harmonic oscillator, plot, and labels.

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

# Create exponentially decaying harmonic oscillator
timearr = np.arange(0.0, 5.0, 0.02)
yarr = np.exp(-timearr) * np.cos(2 * np.pi * timearr)

fig = plt.figure(figsize=(6, 5), facecolor='white')
ax = fig.add_subplot(111)
ax.plot(timearr, yarr, 'r-', markerfacecolor='red', linewidth=3)
plt.xlim(0, 5)
plt.ylim(-1, 1)
ax.grid(True)
ax.set_ylabel('Amplitude')
ax.set_xlabel('Time')
t = plt.title('Damped Harmonic Oscillator')

plt.show()
```

Figure 2.2 shows the result of this Python script. Other examples can be found on the Matplotlib gallery page<sup>8</sup>.

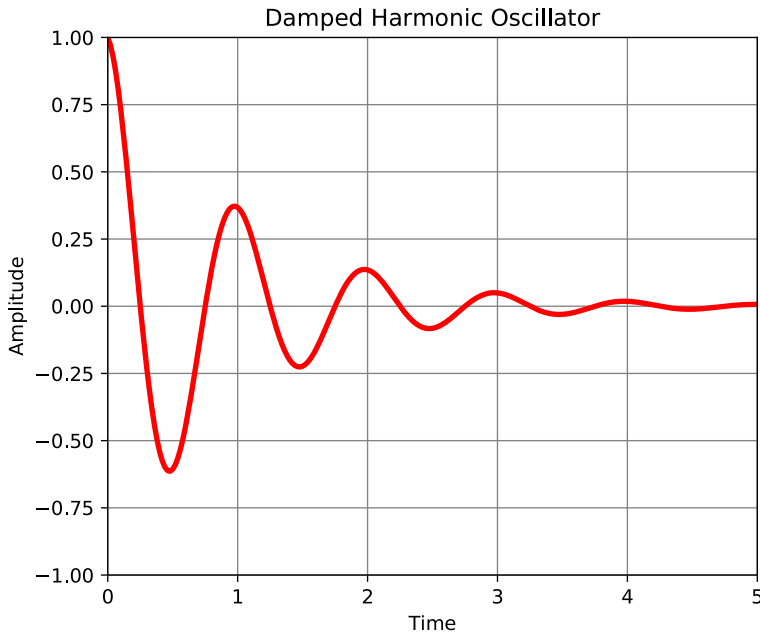
## 2.4 *Mathematica* and *Wolfram Alpha*

*Mathematica*<sup>®</sup> is an extensive symbolic and natural language software package for computation<sup>9</sup> [4]. It is proprietary software and is included with the Raspberry Pi. *Wolfram Alpha* serves as a repository that can be queried with natural language and other mathematical expressions across a wide variety of scientific data. The Wolfram Alpha website allows users to the semantic natural language interface with data

<sup>8</sup><https://matplotlib.org/gallery.html>

<sup>9</sup><http://reference.wolfram.com/language/>





**Figure 2.2.** Example of a simple Matplotlib graph.

products via a web interface or through *Mathematica*, the latter of which can be utilized in this work. The scope of *Mathematica* software and the Wolfram language reaches far beyond this text, so we will cover the elements as needed to explore the exercises outlined in our contained chapters.

*Mathematica* code is stored in notebooks (\*.nb), which are syntax highlighted, annotated, and context sensitive. Command execution are carried out via Functions. Functions use the notation *Function*[ ] with square brackets and a *Capital letter*. Lists of dimension N are defined with curly brackets, and are indexed starting with integer **1** via double square brackets. Input/Output is displayed in adjacent sequential lines unless suppressed with a semicolon. Table 2.1 describes some of this functionality.

Each command is executed with *Shift-Enter*. If output needs to be suppressed, a semicolon (;) should be added to the end of the line. A context sensitive menu of choices will typically appear with the output, guiding the user to options that will allow them to enhance or perform further computations with the output. These menus are incredibly useful for first time users as they act as guidance to previously unexplored functions and *Mathematica* features.

*Mathematica* is sparse with looping mechanisms, and prefers to use mapping and functions to achieve its computations. This often greatly simplifies the code. The **Rule** syntax, indicated by an arrow ( $\rightarrow$ ) transforms a given rule into each part of a given expression. It is often used with the **ReplaceAll** syntax (/.) to evaluate expressions and tabulate results. Appendix A has a basic list of commonly used shortcuts. More can be found on the Wolfram *Mathematica* Documentation website<sup>10</sup>.

<sup>10</sup> <http://reference.wolfram.com/language/>

**Table 2.1.** Commonly used *Mathematica* functions.

<i>Mathematica</i> Functions		
Function	Syntax Example	Description
Wolfram Alpha	=	Semantic natural language processing
Plot	Plot[Cos[x],{x,0,2 Pi}]	Two-dimensional plotting
Quantity	Quantity[20, "Meters"]	Datatype units associated with a numeric value.
UnitConvert	UnitConvert[[20, "Meters"], "Feet"]	Convert between physical units—usually used in concert with Quantity[]
NDSolve	NDSolve[]	Numerically solve ordinary differential equations
List	{1,2,3}	Create a basic array type structure displayed as {1,2,3}
Table	Table[Sin[x],{x,0,2 Pi,Pi/4}]	List generator function

A comprehensive example of function and rule usage is given next. We will create a 3D model of a parameterized Möbius strip. The parameterization is given as

$$x(u, v) = \left(1 + u \cos\left(\frac{v}{2}\right)\right) \cos v \tag{2.1}$$

$$y(u, v) = \left(1 + u \cos\left(\frac{v}{2}\right)\right) \sin v \tag{2.2}$$

$$z(u, v) = u \sin\left(\frac{v}{2}\right). \tag{2.3}$$

With  $-0.5 < u < 0.5$  and  $0 < v < 2\pi$ , we set parameters with **Rules** of Mesh, PlotPoints, Boxed, Axes, Background, PlotRange, PlotStyle, and Lighting.

```
mob = ParametricPlot3D[{(1 + u Cos[0.5 v]) Cos[v],
    (1 + u Cos[0.5 v]) Sin[v],
    u Sin[0.5 v]},
    {u, -0.5, 0.5}, {v, 0, 2 Pi},
    Mesh -> False, PlotPoints -> 20, Boxed ->False,
    Axes -> False, Background -> Black,
    PlotRange -> {{-3, 3}, {-3, 3}, {-3, 3}},
    PlotStyle -> {Specularity[White, 10]},
    Lighting -> {"Directional",
        RGBColor[0.5', 0.5', 1],
        ImageScaled[{0, 1, 0}]},
        {"Directional",
        RGBColor[1, 0.5', 0.5'],
        ImageScaled[{1, -1, 0}]},
```

```

        {"Directional",
         RGBColor[0.5', 1, 0.5'],
         ImageScaled[{-1, -1, 0}]}
    ]

```

We can now use the **ReplaceAll** and **ListAnimate** syntax to control the viewing geometry on the resulting List.

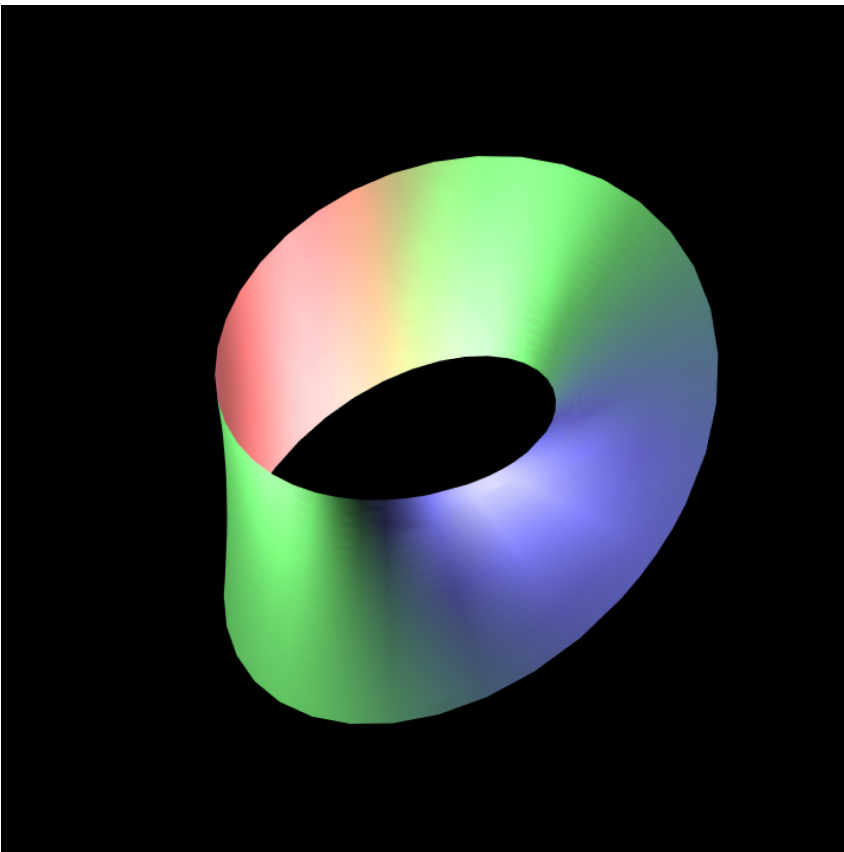
```

s = table[
  Show[mob, ViewAngle -> 20, Background -> Black] /.
    gg : GraphicsComplex[___] :>
      Rotate[gg, theta, {1, 0.5, 0}], {theta, 0., 2.*Pi, 0.1}];

ListAnimate[s]

```

Figure 2.3 shows the result. This can be manipulated in the downloadable notebook mentioned at the end of the chapter.



**Figure 2.3.** 3D model of a Möbius strip used to demonstrate how functions, parameters, rules, and replacements are used in *Mathematica*.

*Mathematica* also interfaces with *Wolfram Alpha*, a computational knowledge engine that can be accessed via the equals sign (=). This semantic computing allows for natural language querying of data and computations. Popup menus will be displayed after a line of notebook output, allowing the user to further explore the result, through visualization, tabulation, or reformatting of the result.

## 2.5 Sources of astronomical science data

Certain exercises in this book will, directly or indirectly through another data provider, use curated data from scientific archives for astronomy and astrophysics. This non-exhaustive list of resources is provided so the reader may experiment with different types of scientific data.

---

<b>Skyview telescope</b>	Multi-wavelength astronomical data [2] <a href="https://skyview.gsfc.nasa.gov">https://skyview.gsfc.nasa.gov</a>
<b>Juno Mission Archive</b>	Ephemeris data and planetary imaging <a href="https://www.missionjuno.swri.edu/junocam/processing">https://www.missionjuno.swri.edu/junocam/processing</a>
<b>NRAO archive</b>	Radio astronomy data and imaging <a href="http://archive.nrao.edu/">http://archive.nrao.edu/</a>
<b>Hubble Legacy Archive</b>	Raw and processed images from HST instruments <a href="http://archive.mast.edu">http://archive.mast.edu</a>

---

## 2.6 Using revision control

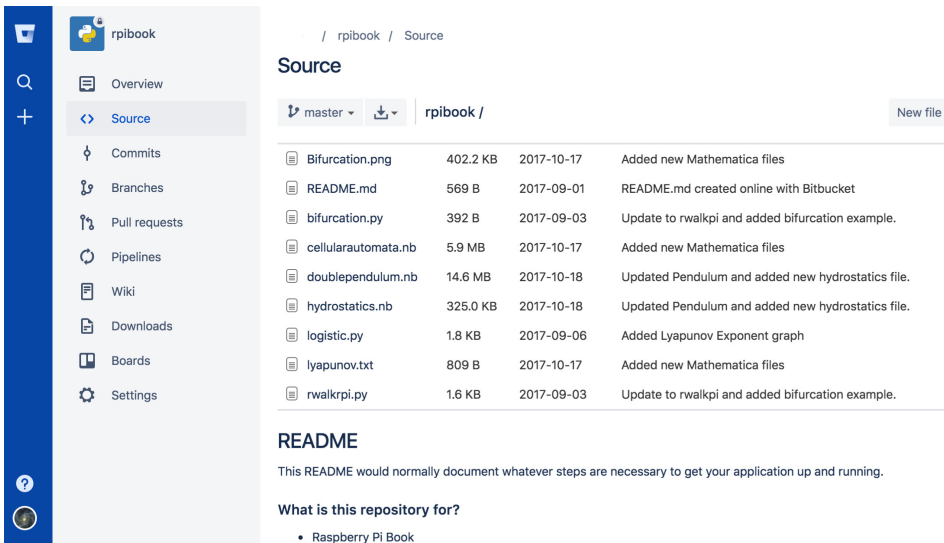
Revision control is used in software development to track and manage changes during code development. While the code exhibited in this work does not have the characteristics of a large project, it does allow the working scientist to keep track of the evolution of their computing experiments. In essence, revision control can be a logbook of your activities as an experiment evolves—it is the modern scientist’s equivalent of a laboratory notebook.

We recommend using *git* as a revision control system<sup>11</sup>. It is extremely versatile and powerful as a means of code management. From an existing directory of data, we can issue the following commands

```
mkdir project
cd project
touch code.py
git init
git add code.py
```

---

<sup>11</sup> <https://git-scm.com/>



**Figure 2.4.** Bitbucket is one of many hosting repositories for source code management using *git*. The GUI interface here gives all the revision history of all code that has been committed to this particular repository.

If we edit the file *code.py*, and then wish to commit and add those changes to our repository,

```
git commit -m "File update"
git push -u origin master
```

An example Bitbucket Git repository is shown in figure 2.4. One important feature of using revision control is the ability to see a history of changes to the code. The *diff* view in a development environment or website like Bitbucket will show where and when changes were made (figure 2.5).

## 2.7 Jupyter notebooks

Jupyter notebooks in Python give similar functionality as *Mathematica* notebooks with interactive content in a web browser<sup>12</sup>. Inline plots can be activated with

```
%matplotlib inline
```

The advantage of using a Jupyter notebook is that code can be shared with rich text and headings in a web page format, and edited and interacted with by a user. Figure 2.6 shows what the interface looks like. The user can also use Jupyter notebooks in the Google Colab interface<sup>13</sup>.

<sup>12</sup> <https://jupyter.org/>

<sup>13</sup> <https://colab.research.google.com>. This allows sharing of Python scripts, using GPUs, and easily importing many of the Python libraries used in this text.

```

40 41
41 42 #-----
42 43 # Bifurcation Curve
43 -ax2 = fig.add_subplot(122)
44 +ax2 = fig.add_subplot(222)
44 45
45 -for alpha in list(np.linspace(2.8,4.0,1000)):
46 +lambdas = []
47 +alphas = list(np.linspace(2.8,4.0,1000))
48 +for alpha in alphas:
46 49     Xn = 0.1
47 50     Xnvals = [Xn]
48 -    print 'Alpha:', alpha
51 +    lyapunovvals = []
49 52     for i in range(iterations):
50 53         Xnp1 = alpha * Xn * (1 - Xn)
51 54 +        lyapunovvals.append(log(abs(alpha - 2 * alpha * Xn)))
51 55         Xn = Xnp1
52 56         Xnvals.append(Xnp1)
57 +        lambdas.append(np.mean(lyapunovvals))
58 +
59 +    print 'Alpha:', alpha, "    Lambda:", np.mean(lyapunovvals)
53 60
54 61     Xnvalsplot = Xnvals[len(Xnvals)/2:-1]
55 62     ax2.scatter(np.repeat(alpha,len(Xnvalsplot)),Xnvalsplot, s=0.01, color='red')

```

**Figure 2.5.** Bitbucket is one of many hosting repositories for source code management using *git*. The GUI interface here gives all the revision history of all code that has been committed to this particular repository.

## 2.8 Coding pedagogy

Each chapter in this book is broken down into sections that will cover a particular exercise. We will introduce the scientific or technical nature of the exercise, what software and libraries will be used, how the exercise is carried out, and how results are obtained and displayed. Some sections will use Python and some *Mathematica*. We will illustrate the strengths of each programming system and language via the presented exercises. Each section will contain a list of requirements and objectives for the exercise, as well as a brief written algorithm to outline what we wish to accomplish. As part of good coding practice and scientific inquiry, such a procedure should become standard practice in all software exercises.

Online content and examples for this book can be accessed at

<https://bitbucket.org/brkent/raspberrypi>

The git repository and code can be cloned at the Linux command prompt with

```
git clone https://bitbucket.org/brkent/raspberrypi.git
```

### Python exercises

1. Use the Python datetime module to format today's day, date, and time.
2. Create a Python function to convert temperature from degrees Fahrenheit to Celsius.

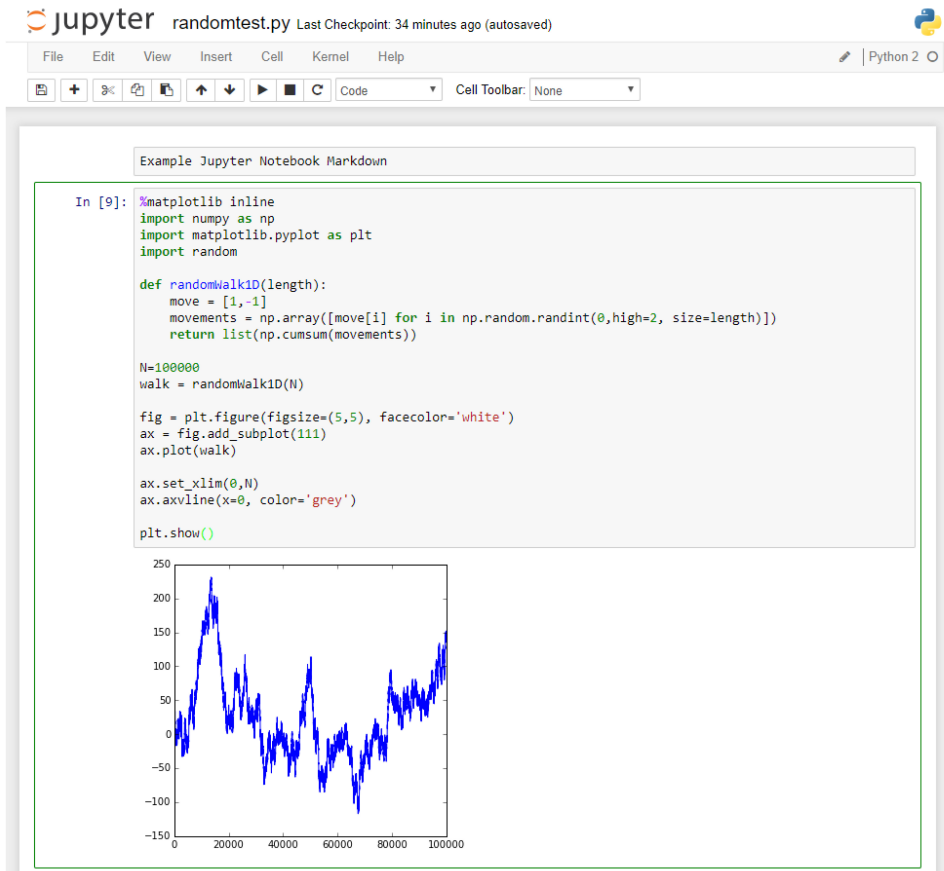


Figure 2.6. Example of a Jupyter notebook.

3. Create a Python function to search a list of integers and the number of occurrences of each integer.
4. Determine a list of installed Python modules with your current installation.
5. Create a repository with Bitbucket or GitHub and commit the code you just created.

### **Mathematica exercises**

1. Create a new *Mathematica* notebook, and obtain basic data with the Wolfram Alpha interface.
2. Generate a two-dimensional matrix (list) using the **Table** function.
3. Use **ListAnimate** or **Manipulate** to change the values of the matrix.

## References

- [1] Kent B R 2015 *3D Scientific Visualization with Blender* (Bristol: IOP Publishing)
- [2] McGlynn T, Scollick K and White N 1998 SKYVIEW: The multi-wavelength sky on the Internet In *New Horizons from Multi-Wavelength Sky Surveys*; McLean B J, Golombek D A, Hayes J J E and Payne H E 465 *Proc. 179th Symp. Int. Astron. Union*
- [3] Pilgrim M 2009 *Dive Into Python 3* (Berkeley, CA: Apress)
- [4] Wolfram S 2017 *An Elementary Introduction to the Wolfram Language* (Hanborough: Wolfram Media)



# Chapter 3

## Chaos and non-linear dynamics

### 3.1 One and two dimensional pseudo random walks

As a useful exercise to start understanding how to write some basic Python code with an interesting result, we will begin by generating one and two-dimensional pseudo random walks. This exercise will serve a number of purposes—(i) Show basic Python module usage, (ii) create an important example in generating random samples for experiments, and (iii) graphing the results with Matplotlib.

#### Goals

- Write a first Python program.
- Understand the usage of Python modules and graphics packages.
- Generate and analyze a pseudo random walk.
- Start at the origin  $(x,y) = (0,0)$ .
- Roll the dice and move in one of four directions randomly for each step—up, down, left, or right.
- Repeat for a specified number of  $N$  steps.
- Values will be returned as a list of  $x$  and  $y$  positions for each step.

#### Code description and algorithm

The libraries we will need consist of

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

We assume that the walk begins at the origin in Cartesian coordinates. Our movement is limited to one integer step forward or backward, essentially flipping a fair coin each time with two possible outcomes.

```
def randomWalk1D(length):
    move = [1,-1]
    movements = np.array([move[i]
                          for i in
                          np.random.randint(0,high=2, size=length)])
    return list(np.cumsum(movements))
```

We will create a walk of 100 000 steps, plot the results, and save the figure.

```
# Create the walk
N = 100000
walk = randomWalk1D(N)

# Plot the results with matplotlib
fig = plt.figure(figsize=(10,10), facecolor='white')
ax = fig.add_subplot(111)
ax.plot(walk)

ax.set_xlim(0,N)
ax.axvline(x=0, color='grey')

ax.set_xlabel('Step')
ax.set_ylabel('Position from origin')
ax.set_title('1D Pseudorandom Walk')

plt.savefig('rwalk1D.png')

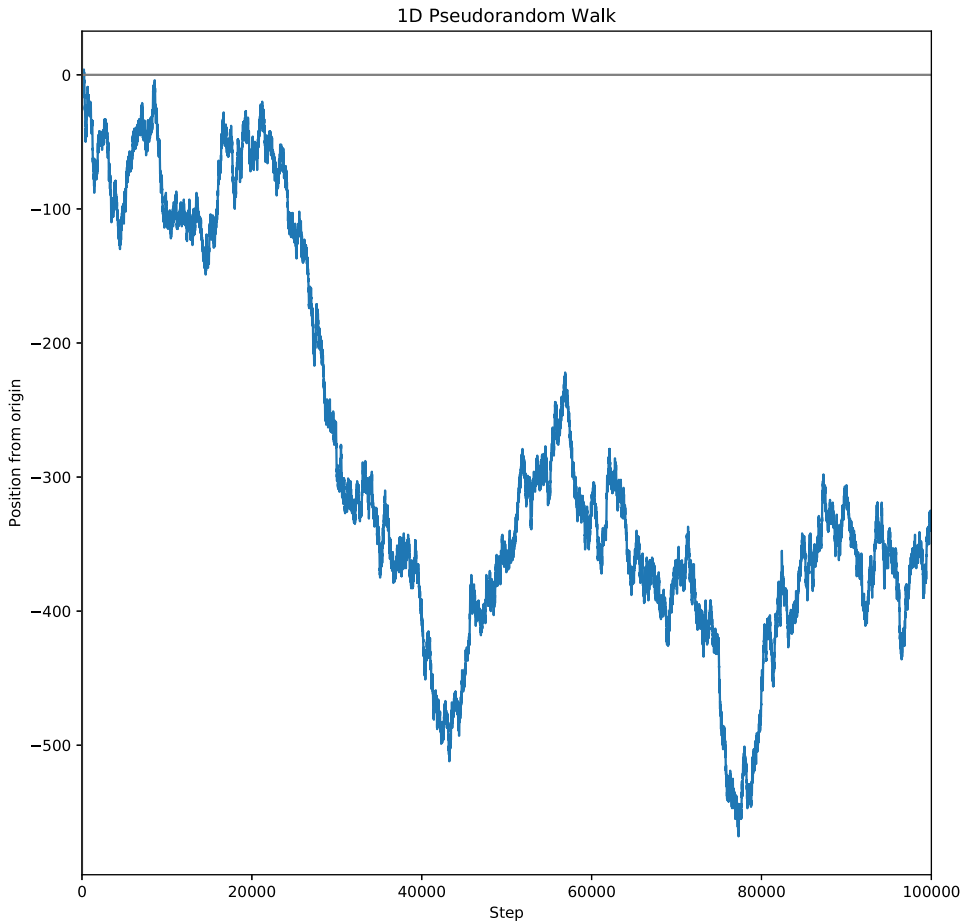
plt.show()
```

We note that the **savefig** method chooses the output based on the file suffix. A publication quality EPS or PDF file can also be obtained. Figure 3.1 shows the result of this code.

A two-dimensional pseudo random walk function can be written in a more concise manner, but for now we will write it out explicitly such that it is clear to the reader. It expands upon the 1D example but uses color coding to visualize the steps.

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

def randomWalk2D(length):
    x,y = 0,0
    walkx,walky = [x],[y]
    for i in range(length):
```



**Figure 3.1.** A one-dimensional pseudo random walk. The graph shows the displacement from the origin along the number line for 100 000 steps. Each step allows a movement of an integer value of 1 or  $-1$  along the  $y$ -axis.

```

new = random.randint(1, 4)
if new == 1:
    x += 1
elif new == 2:
    y += 1
elif new == 3:
    x += -1
else:
    y += -1
walkx.append(x)
walky.append(y)
return [walkx,walky]

```

The main code block iterating over 10 000 steps with graphic elements is shown below. At each step, the cursor can move an integer value of one up, down, left, or right.

```
N = 10000
walka = randomWalk2D(N)

fig = plt.figure(figsize=(10,8), facecolor='white')
ax = fig.add_subplot(111)

im = ax.scatter(walka[0],walka[1],marker='s',
               linewidth=0,s=15,c=range(N+1))

ax.axhline(linewidth=1, color='blue', linestyle='--')
ax.axvline(linewidth=1, color='blue', linestyle='--')
ax.axhline(y=walka[1][-1], linewidth=1, color='red', linestyle='--')
ax.axvline(x=walka[0][-1], linewidth=1, color='red', linestyle='--')
ax.axis('equal')

# Colorbar
cb = fig.colorbar(im, ax=ax)
cb.set_label('Steps')

# Set Labels
ax.set_xlabel('X position')
ax.set_ylabel('Y position')
ax.set_title('2D Random Walk')

plt.savefig('randomwalk2D.png')

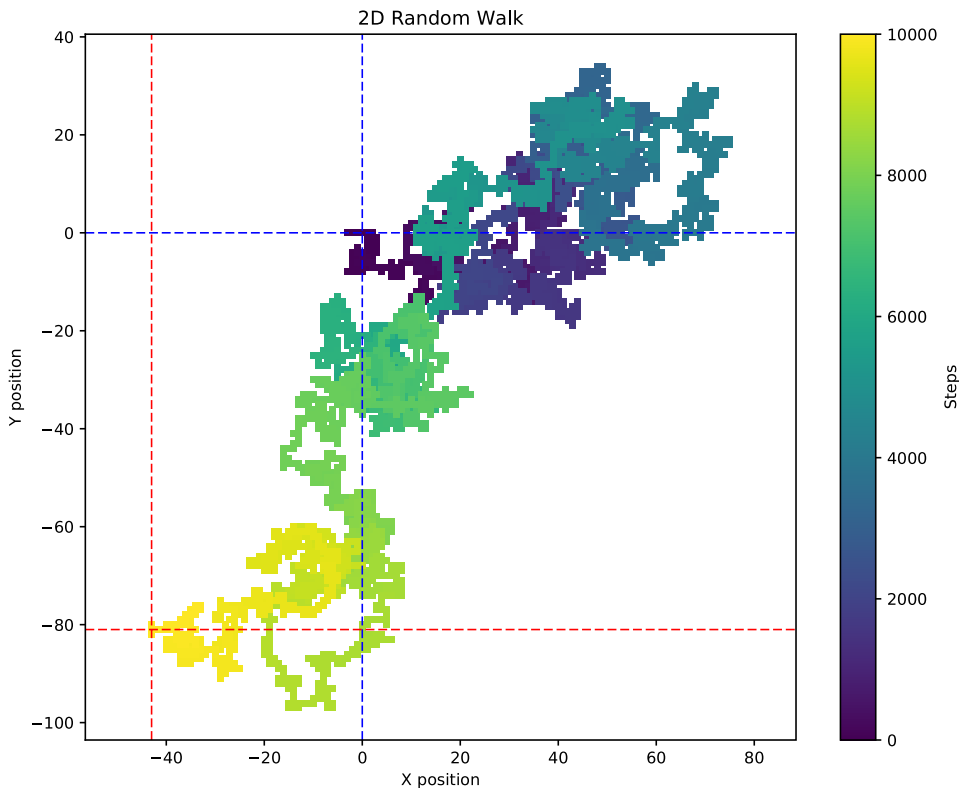
plt.show()
```

Figure 3.2 depicts blue dashed lines showing the start of the random walk, with dark red dashed lines indicating the finish.

### Where does a random walk end up?

We can create many instances of our random walk over and over again, and examine the distribution of where the random walk ends up finishing after  $N$  steps [3]. Using the same **randomWalk2D** function

```
N = 1000
finalx=[]
finaly=[]
Ntrials = 10000
for i in range(Ntrials):
    walk=randomWalk2D(N)
    finalx.append(walk[0][-1])
    finaly.append(walk[1][-1])
```



**Figure 3.2.** A two-dimensional pseudo random walk. In this graph, the evolution of steps is color coded from dark blue to yellow for 10 000 steps. Dashed blue and red lines show the start and ending points of the sequence respectively.

```

dist = []
for i in range(0, len(finalx)):
    dist.append(math.sqrt(finalx[i]**2 + finaly[i]**2))

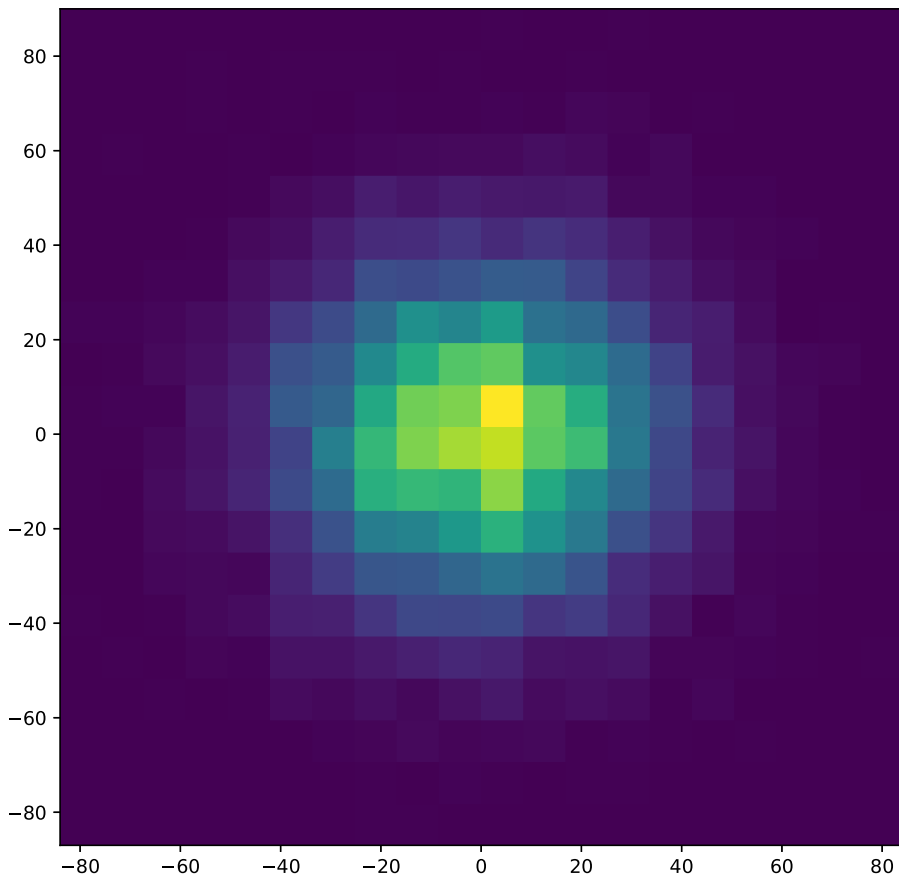
fig = plt.figure(figsize=(8,8), facecolor='white')
ax = fig.add_subplot(111)
im = ax.hist2d(finalx, finaly, bins=20)
plt.savefig('randomwalk2Danalysis.png')
plt.show()

```

We have completed 10 000 trials of 1000 steps and binned the results into a two-dimensional histogram, shown in figure 3.3.

### Exercises.

1. Make a histogram of the 1D pseudo random walk and show the distribution of where multiple trails end up after  $N$  steps.



**Figure 3.3.** 10 000 trials of a 2D pseudo random walk, each with 1000 steps. The resulting ending points of each trial have been binned in  $10 \times 10$  grid elements to make a 2D histogram.

2. Determine the root mean square distance  $\sqrt{N}$  of the 2D pseudo random walk and plot versus  $N$  steps during the progression of the walk.
3. Keeping the number  $N$  steps small ( $N = 100$ ), write a new function to make a 3D pseudo random walk.

## 3.2 Logistic maps, bifurcation, and chaos

### Goals

- Understand how to create a logistic mapping function and what it represents.
- Create a non-linear recurrence relation.
- Created a cobweb plot in Python.
- Show convergence (or lack thereof) for the different parameter values and using the same calculation, compute a bifurcation plot.
- Identify chaos in the non-linear relation.

- Compute values for several hundred iterations with different parameter values.
- Determine the derivative and compute the Lyapunov exponent.

### Code description and algorithm

Non-linear dynamics and chaotic phenomena appear in many places in nature. Many chaotic systems exhibit elegant mathematics that create incredible patterns of stunning complexity. The initial conditions of a dynamic system can greatly impact the temporal evolution of state. Physical phenomena can be modeled with a discrete difference equation such that

$$x_{n+1} = f(\alpha, x_n). \quad (3.1)$$

Based on input parameters  $\alpha$  and  $x_n$ , the result is a *mapping* of the function. Often these recurrence relations provide useful numerical solutions to non-linear equations. The *logistic map* represents a classic example with a one-dimensional equation [5]

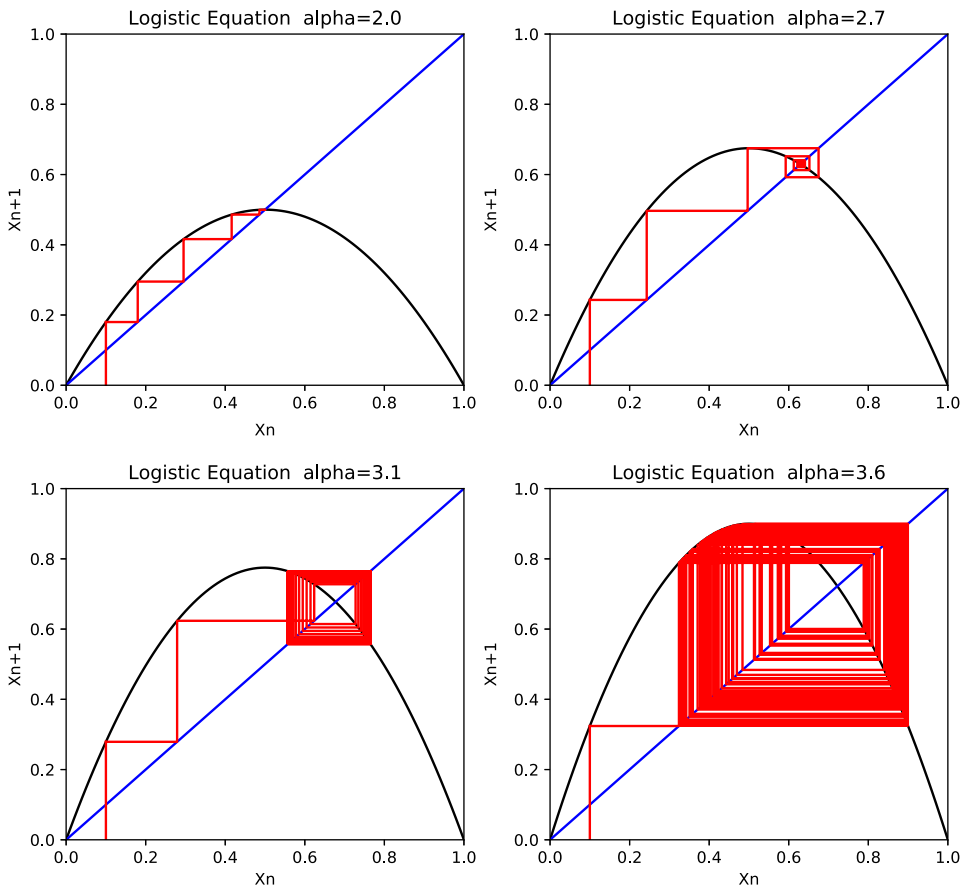
$$f(\alpha, x_n) = x_{n+1} = \alpha x_n(1 - x_n). \quad (3.2)$$

It is easiest to compute the values to see how  $x_n$  propagates to  $x_{n+1}$  for a given value of parameter  $\alpha$ . Computed values for this mapping are shown in table 3.1.

We visualize the computed values in table 3.1 with a curve of the quadratic function and intersecting the  $\pi/4$  radians one-to-one relation line. For a value of  $\alpha = 2.7$ , the relation converges to a single value of 0.630. For other values of  $\alpha$ , (i) convergence takes a larger number of iterations, (ii) a periodic relation is developed oscillating between two values ( $\alpha = 3.1$ ), or (iii) a sensitive dependence on initial conditions results in chaotic behavior that never converges to a single value or two ( $\alpha = 3.6$ ). Moving from  $x_1$  to  $x_n$  is nicely shown in a cobweb plot for several values of  $\alpha$  (figure 3.4).

**Table 3.1.** Logistic map values for  $\alpha = 2.7$  showing convergence and  $\alpha = 3.1$  showing oscillation between two values.

Logistic Map Values		
$n$	$\alpha = 2.7$	$\alpha = 3.1$
0	0.100	0.100
1	0.243	0.279
2	0.497	0.623
3	0.675	0.728
4	0.592	0.614
5	0.652	0.734
...	...	...
97	0.630	0.765
98	0.630	0.558
99	0.630	0.765



**Figure 3.4.** Diagrams known as cobweb plots are generated for the Logistic Map equation. The top panels show  $\alpha$  values show fast convergence ( $\alpha = 2.0$ ) and slower convergence ( $\alpha = 2.7$ ) to a single value. The bottom panels show  $\alpha$  values with an oscillating return between two values ( $\alpha = 3.1$ ) and chaotic behavior that never converges ( $\alpha = 3.6$ ). The blue line shows a one-to-one correspondence, and the black parabola indicates possible values of the recurrence relation.

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

fig = plt.figure(figsize=(4.5,4), facecolor='white')
ax1 = fig.add_subplot(111)

iterations = 100

x = np.linspace(0, 1, 101)
alpha = 2.7
y = alpha * x * (1 - x)
```



```

ax1.plot(x,y, color='black')
ax1.plot(x,x, color='blue')

plt.xlim(0.0,1.0)
plt.ylim(0.0,1.0)

# Plot vector of path from Xn to Xn+1
Xn = 0.1
yval = 0.0
Xnvals = [Xn]
for i in range(iterations):
    Xnp1 = alpha * Xn * (1 - Xn)
    ax1.plot([Xn,Xn],[yval,Xnp1], color='red')
    ax1.plot([Xn,Xnp1], [Xnp1,Xnp1], color='red')

    yval = Xnp1
    print(i, Xn)
    Xn = Xnp1

    Xnvals.append(Xnp1)

ax1.set_xlabel('Xn')
ax1.set_ylabel('Xn+1')
ax1.set_title('Logistic equation alpha={!s}'.format(str(alpha)))

plt.savefig('logisticAlpha{!s}.png'.format(alpha))

plt.show()

```

If we define an arbitrary maximum number of iterations  $N = 1000$  for each value of  $\alpha$ , convergence is reached by  $N/2$  iterations. Plotting  $x_n$  versus  $\alpha$  for all points where the quadratic function is intersected, we obtain a fascinating bifurcation diagram (figure 3.5). This clearly shows where stability in the non-linear relation is obtained.

```

import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(7,5), facecolor='white')

iterations = 1000

# Bifurcation Curve
ax2 = fig.add_subplot(111)

lambdas = []

```

```

alphas = list(np.linspace(2.8,4.0,iterations))
for alpha in alphas:
    Xn = 0.1
    Xnvals = [Xn]
    for i in range(iterations):
        Xnp1 = alpha * Xn * (1 - Xn)
        Xn = Xnp1
        Xnvals.append(Xnp1)
    Xnvalsplot = Xnvals[len(Xnvals)/2:-1]
    ax2.scatter(np.repeat(alpha,len(Xnvalsplot)),
                Xnvalsplot, s=0.005, color='red')

ax2.axvline(x=3.1, linewidth=1, color='black', linestyle='-')
ax2.axvline(x=3.6, linewidth=1, color='black', linestyle='-')

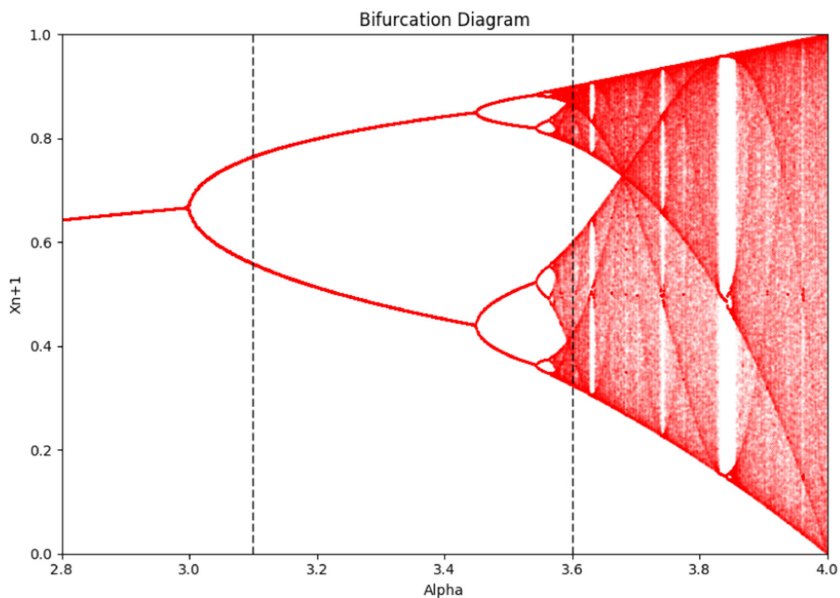
plt.xlim(2.8,4.0)
plt.ylim(0.0,1.0)

ax2.set_xlabel('Alpha')
ax2.set_ylabel('Xn+1')
ax2.set_title('Bifurcation Diagram')

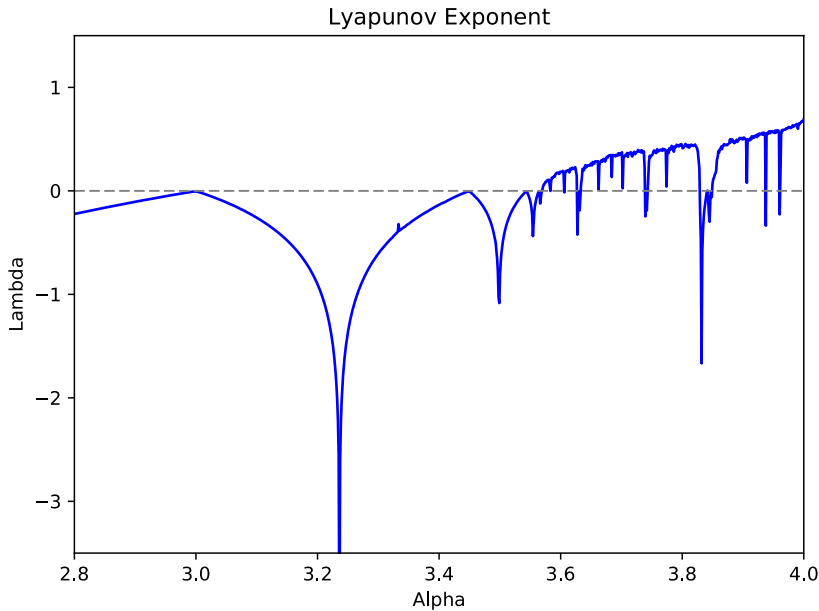
plt.savefig('bifurcation.png')

plt.show()

```



**Figure 3.5.** A bifurcation diagram of the logistic map equation. This graph is generated by plotting values after 100 iterations. For values of  $\alpha < 3.0$ , the recurrence relation converges to a single value. For  $3.0 < \alpha \lesssim 3.45$ , the relation oscillates between two values. For  $3.45 \lesssim \alpha \lesssim 3.55$ , the oscillations occurs between four values. After that, the behavior becomes chaotic, albeit with islands of stability.



**Figure 3.6.** Computing the Lyapunov exponent gives a quantitative measure of chaos. For values of  $\lambda > 0$ , the recurrence relation does not converge and is chaotic. The steep depressions show regions of stability, and areas where there is a turnover at the zero line show a region where the oscillations split.

We can identify chaotic behavior in our experiment via the Lyapunov characteristic exponent [4]. The value quantifies the average exponential growth per unit time between two states. If we define the difference between successive iterations as

$$\Delta X_n = \epsilon e^{n\lambda}. \quad (3.3)$$

We can easily evaluate this in our code—simply differentiate our logistic map function and average over  $N$  iterations. In the limit that  $n$  goes to infinity, we define the Lyapunov exponent with

$$\lambda = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} \ln \left| \frac{df(x_i)}{dx} \right|. \quad (3.4)$$

Figure 3.6 shows the Lyapunov exponent  $\lambda$  versus  $\alpha$ . For  $\alpha > 3.57$ , chaotic behavior ( $\lambda > 0$ ) is exhibited, as are windows of periodicity or convergence ( $\lambda < 0$ ).

```
import numpy as np
import matplotlib.pyplot as plt
from math import log

fig = plt.figure(figsize=(7,5), facecolor='white')

iterations = 1000
```

```

# Lyapunov Exponent Calculation
lambdas = []
alphas = list(np.linspace(2.8,4.0,iterations))
for alpha in alphas:
    Xn = 0.1
    Xnvals = [Xn]
    lyapunovvals = []
    for i in range(iterations):
        Xnp1 = alpha * Xn * (1 - Xn)
        lyapunovvals.append(log(abs(alpha - 2 * alpha * Xn)))
        Xn = Xnp1
        Xnvals.append(Xnp1)
    lambdas.append(np.mean(lyapunovvals))

    outputstring = 'Alpha: {0:.3f} Lambda: {1:.3f}'
    print(outputstring.format(alpha, np.mean(lyapunovvals)))

    Xnvalsplot = Xnvals[len(Xnvals)/2:-1]

# Plot Lyapunov Exponent versus Alpha
ax3 = fig.add_subplot(111)

ax3.plot(alphas, lambdas, color='blue')
ax3.axhline(y=0.0, linewidth=1, color='grey', linestyle='-')
plt.xlim(2.8,4.0)
plt.ylim(-3.5,1.5)
ax3.set_xlabel('Alpha')
ax3.set_ylabel('Lambda')
ax3.set_title('Lyapunov Exponent')

plt.savefig('lya.png')

plt.show()

```

### Exercises

1. Determine and plot the bifurcation diagram for negative values of  $\alpha$ . Is the appearance of chaotic behavior symmetric with the positive side of the  $x$ -axis?
2. Compute and plot the Lyapunov exponent lambda for negative values of  $\alpha$ .

## 3.3 Cellular automata

### Goals

- Create your first *Mathematica* notebook.
- Understand the usage of *Mathematica* syntax, functions, and data structures.
- Create cellular automata diagrams.

- Cells in a simple one-dimensional cellular automaton have a binary state, *on* or *off*.
- The neighborhood of a cell—the cell itself, and the cells immediately adjacent to the left and right are grouped together to determine the next step in the automaton evolution.
- The determination of this outcome is called a rule, often identified as the Wolfram number.

### Algorithm and code description

A cellular automaton is a discrete numerical system typically represented by the following criteria.

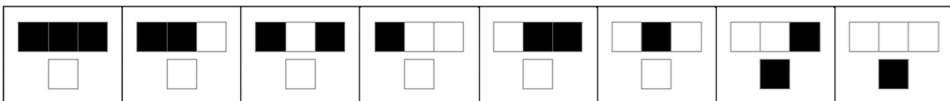
- A lattice grid of arbitrary dimension showing a visual representation of the state of the automaton.
- A specified set of rules dictating the propagation of a state  $n + 1$  based on the state  $n$ .

The lattice grid will evolve based on the set of numerical rules of state, with outcomes that include symmetric, homogeneous, chaotic, pseudo-random, non-linear, or oscillating behavior. The computing language *Mathematica* is included with Raspberry Pi and provides an innovative platform for doing cellular automata computations. Some automata are mathematical curiosities, while others clearly shown patterns that manifest themselves in nature [2].

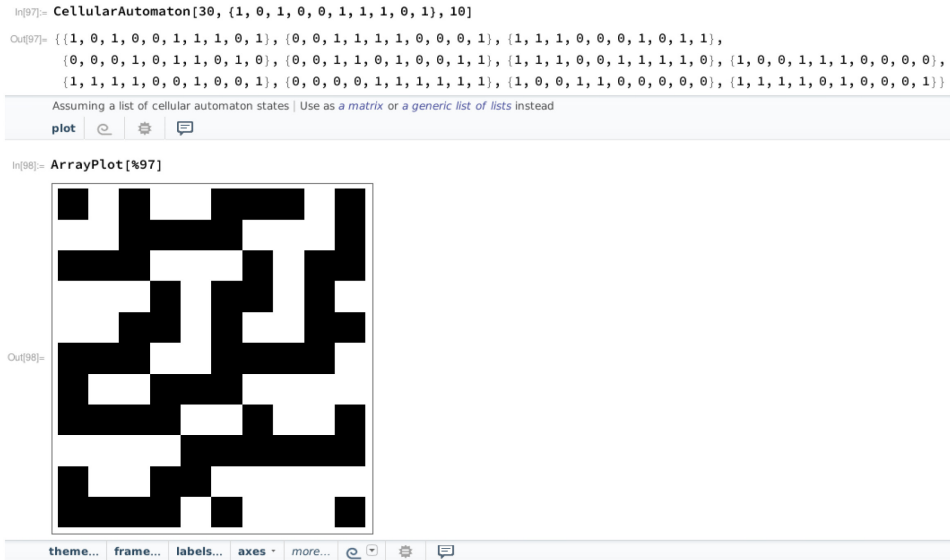
One dimensional elementary cellular automata consist of an orthogonal Cartesian grid where a single grid element state is determined by the previous adjacent grid and the two left/right neighbors (figure 3.7). This triplet pattern has  $2^{2^3} = 256$  possible rule determinations, which are nominally identified by their *Wolfram number*. Many reach a steady homogeneous state quickly, but others exhibit fascinating complex behavior. These automata can be the basis for a number of numerical and physical phenomena. Simply click on the *Mathematica* logo in the upper left corner of your Raspbian desktop to start the program with a new notebook. Code for displaying a cellular automata rule is given by

```
result = CellularAutomaton[3];
RulePlot[result]
```

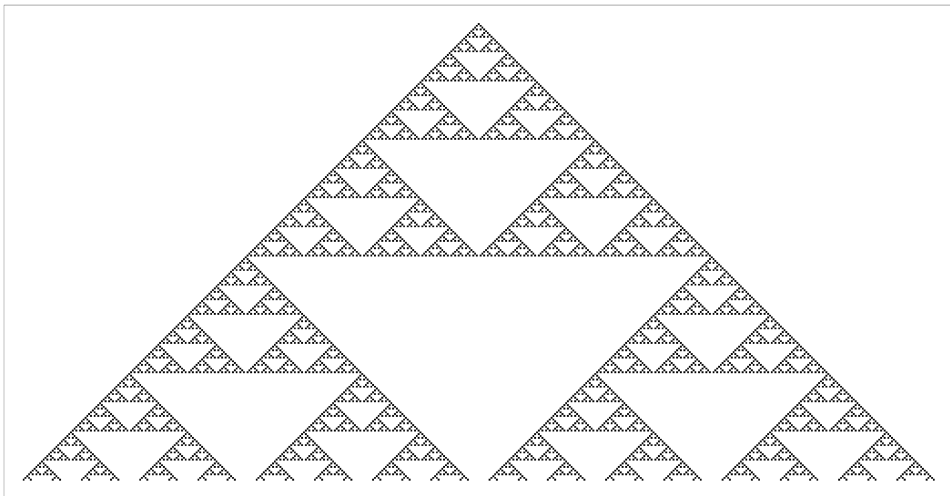
With a new input line in the same *Mathematica* notebook, we can use the **CellularAutomaton** function to create a 1D automaton. This function takes three inputs—a rule, an initial condition (typically a list denoted with curly brackets), and a number of iterations. We can recreate Sierpinski's Triangle with [7]



**Figure 3.7.** Using the *Mathematica* function **RulePlot** for Rule 3 we can see all possible outcomes of a particular rule set where each block has a binary state (0 or 1, black or white).



**Figure 3.8.** After creating a sequence of ten steps with the function `CellularAutomaton`, the *Mathematica* interface will ask the user if they wish to create a plot.



**Figure 3.9.** Cellular automata of Rule 90. This shows the formation of Sierpinski's Triangle.

```
result90 = CellularAutomaton[90, {{1}, 0}, 50];
ArrayPlot[result90]
```

Note that the resulting nested list is stored in the variable `result`, and the screen output is suppressed by the semicolon. The *Mathematica* notebook interface is sophisticated enough that it will then ask if we wish to plot the results (figure 3.8). By clicking on **plot** we can see a graphical representation of the result (figure 3.9).



**Figure 3.10.** Conway's Game of Life totalistic two-dimensional cellular automata.

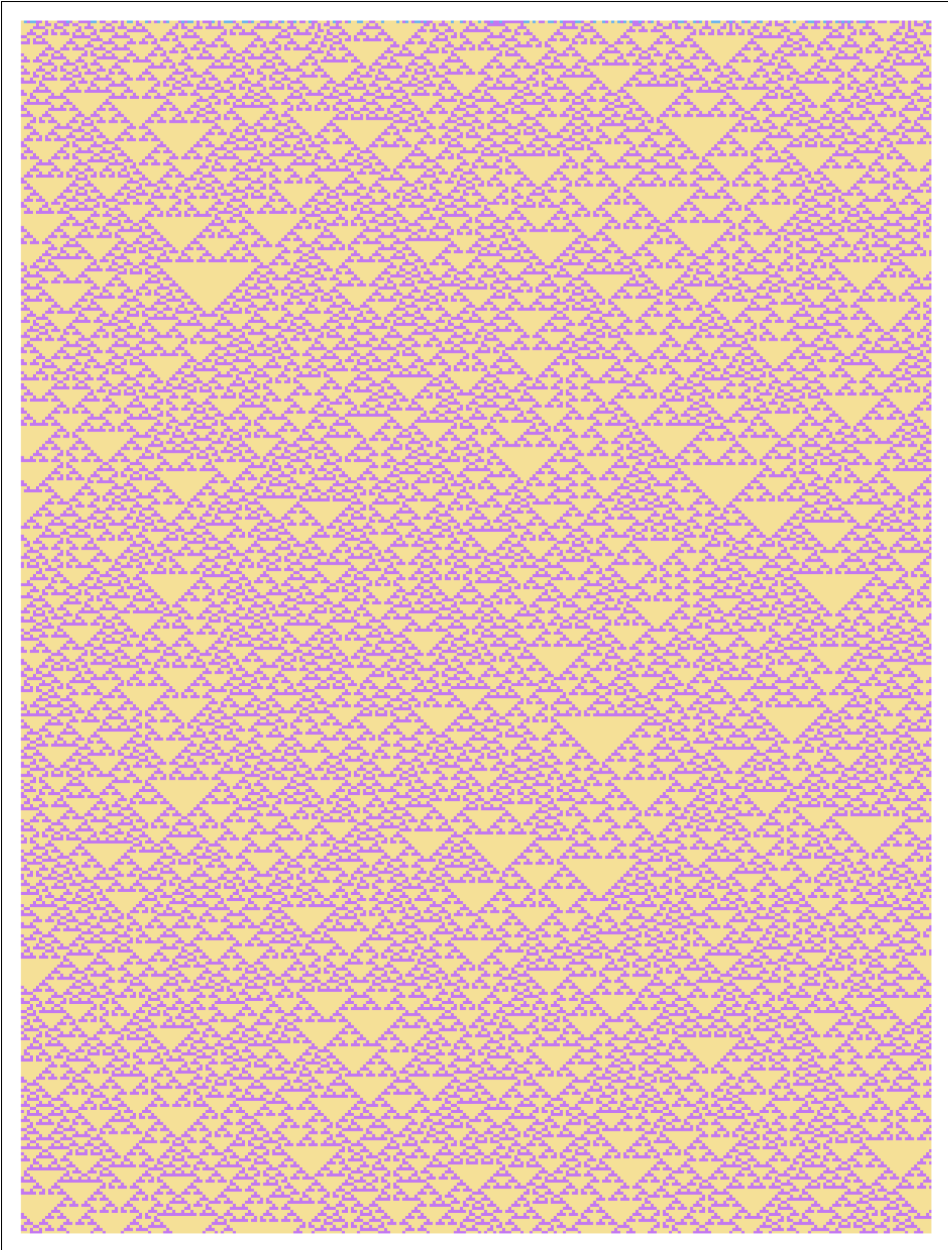
Totalistic cellular automata and two-dimensional automata systems like Conway's Game of Life show how a complicated dynamical system can evolve [1, 6] (figure 3.10). One dimensional totalistic cellular automata are typically colored to distinguished the averaged state from one generation to the next (figure 3.11).

```
ArrayPlot[CellularAutomaton[{1084, {3, 1}, 1}, SeedRandom[42];
                    RandomInteger[2, 401], {400, {-150, 150}}],
          Frame -> True,
          ColorFunction -> "Pastel"]
```

### Exercises

1. The initial condition for Sierpinski's Triangle (Rule 90) create a symmetric geometry as the automaton propagates. Do the same exercise with pseudo random initial conditions. Try the same exercise with a different rule (Rule 110, Rule 60, etc).
2. *Mathematica* supports totalistic multi-state cellular automata calculations. Rather than a square being a simple binary result of 0 or 1 ( $k = 2$ ), one can assign a state to be 0, 1, or 2 ( $k = 3$ ) and color accordingly. How many how many outcomes can there be for a given rule? How many possible rule sets? Using the same CellularAutomaton function, create a plot of one of these rules (there are many choices!).
3. Experiment with Conway's Game of Life with the short Wolfram implementation<sup>1</sup>. This online demo shows that a complex example can be implemented with very little code.

<sup>1</sup><https://www.wolfram.com/language/gallery/implement-conways-game-of-life/>



**Figure 3.11.** Color totalistic cellular automata for 400 steps, random initial conditions and a horizontal spread of 300 pixels.



## References

- [1] Gardner M 1970 The fantastic combinations of John Conway's new solitaire game 'life' *Sci. Am.* **223** 120–3
- [2] Kutrib Martin 2011 Nature-based problems in cellular automata. In ed Benedikt Löwe, Dag Normann, Ivan Soskova and Alexandra Soskova *Models of Computation in Context* (Berlin: Springer) pp 171–80
- [3] Langtangen H P 2016 *A Primer on Scientific Programming with Python* (Berlin: Springer)
- [4] Lyapunov A M 1992 The general problem of the stability of motion *Int. J. Control.* **55** 531–4
- [5] May R 1976 Simple mathematical models with very complicated dynamics *Nature* **261** 459–67
- [6] Packard Norman H and Wolfram Stephen 1985 Two-dimensional cellular automata *J. Stat. Phys.* **38** 901–46
- [7] Wolfram S 1983 Statistical mechanics of cellular automata *Rev. Mod. Phys.* **55** 601–44

# Chapter 4

## Physics and astronomy

### 4.1 A simple pendulum

We can examine the physics of a simple plane pendulum with *Mathematica*.

#### Goals

- Import packages into *Mathematica*.
- Understand the differential equation for a simple pendulum.
- Understand how to solve differential equations in *Mathematica*.
- Make assumptions about the motion for small angles.
- Solve the time dependent differential equation numerically.
- Understand the relationships between the physical parameters of the system.
- Plot the angle and angular velocity as functions of time.

#### Algorithm and code description

The differential equation for this dynamic system with a mass  $m$  constrained to move along a circle of radius  $l$  is given by

$$\ddot{\theta} = -\frac{g}{l} \sin \theta \quad (4.1)$$

where the angle  $\theta$  is a function of time. Like many introductory exercises, we assume that the pendulum will move in small angles such that we can use a linear approximation for the trigonometric sine function

$$\sin \theta \cong \theta \quad (\text{for small angles}). \quad (4.2)$$

This allows us to treat the system as a simple harmonic oscillator, with a solution  $\theta(t)$ , angular frequency  $\omega_0$ , period  $\tau$ , and energy budget of kinetic  $T$  and potential  $V$ , given as

$$\theta(t) = \theta_0 \cos(\omega_0 t) \quad \text{where} \quad \omega_0^2 \equiv \frac{g}{l} \quad (4.3)$$

$$\tau \simeq 2\pi\sqrt{\frac{l}{g}} \quad (4.4)$$

$$T = \frac{1}{2}I\omega^2 = \frac{1}{2} m l^2 \dot{\theta}^2 \quad (4.5)$$

$$V = mgl (1 - \cos \theta). \quad (4.6)$$

This small angle scenario with no dissipation of energy is conservative such that the sum of kinetic and potential energy remain constant. For small angles, the period does not depend on  $\theta_0$ . The *Mathematica* code is very straight forward and intuitive. All physical units are assumed to be SI.

```
theta''[t] == (-g/l)*Sin[theta[t]];

g = 9.8;

l = 2.0;

theta = First[
  theta /. NDSolve[{theta[t] == -(g/l)*Sin[theta[t]],
    theta[0] == Pi/2, theta'[0] == 0}, theta, {t, 0, 10}]];

p1 = Plot[theta[t], {t, 0, 10}, Filling -> Axis, PlotStyle -> Red,
  FrameLabel -> {"t (seconds)", "theta (radians)"}, Frame -> True];

p2 = Plot[theta'[t], {t, 0, 10}, Filling ->Axis, PlotStyle ->Blue,
  FrameLabel ->{"t (seconds)", "theta' (radians/sec)"},
  Frame -> True];

GraphicsColumn[{p1, p2}]
```

Figure 4.1 shows the angle  $\theta$  and angular velocity of this simple pendulum exercise.

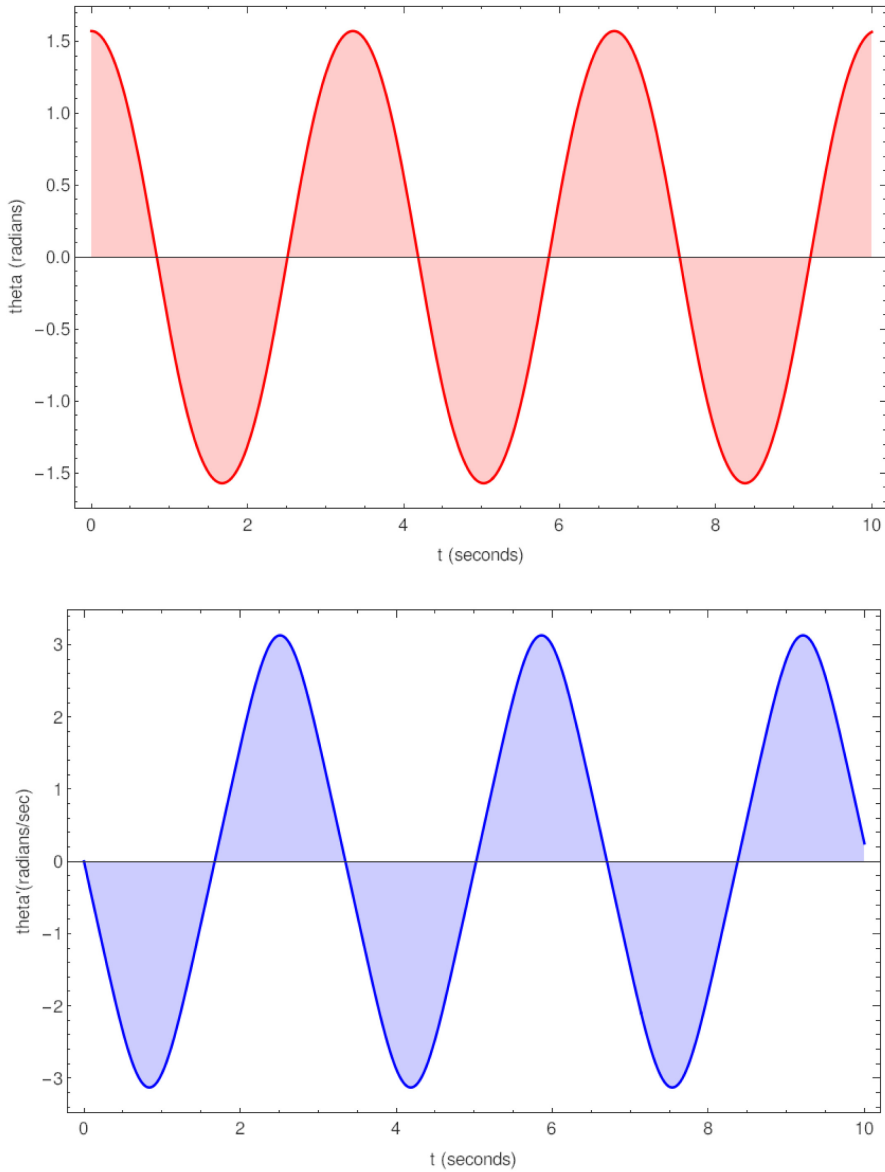
## 4.2 The double pendulum

### Goals

- Understand the Lagrangian of the double pendulum system and solve numerically.
- Use Wolfram Demonstrations to create and study an animation of the time evolution of the double pendulum.

### Algorithm and code description

The double pendulum is an interesting dynamical system that can exhibit chaotic behavior with sensitive dependence on initial conditions. Here we consider two masses each of mass  $m$ , connected by massless rigid rods both of length  $l$  (figure 4.2).

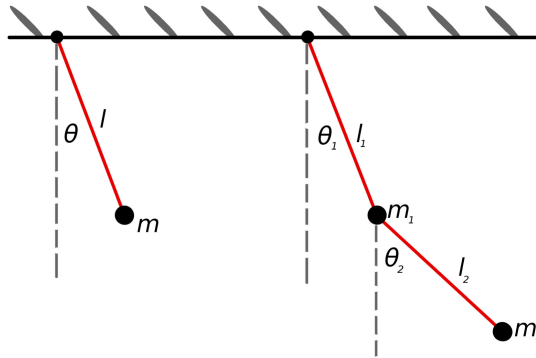


**Figure 4.1.** Solving the simple pendulum motion for small angles and no dissipation of energy. The maximum angular velocity is reached as the pendulum swings through its lowest point at  $\theta = 0$  radians.

The equations of motion for this system are best solved using a Lagrangian formulation  $L$  with the difference of kinetic and potential energies

$$L = T - V. \tag{4.7}$$

We consider generalized coordinates  $\theta_1$  and  $\theta_2$ , massless rod lengths  $l_1$  and  $l_2$ , and masses  $m_1$  and  $m_2$  to get [2]



**Figure 4.2.** The setup of the simple and double pendulum. Each is fixed on a frictionless pivot and massless rods. The introductory computation for the simple pendulum is made for small angles of  $\theta$  whereas the double pendulum can swing a full  $2\pi$  radians.

$$\begin{aligned}
 L = & \frac{1}{2}(m_1 + m_2) l_1^2 \theta_1'(t)^2 \\
 & + \frac{1}{2}m_2 l_2^2 \theta_2'(t)^2 \\
 & + m_2 l_1 l_2 \theta_1'(t) \theta_2'(t) \cos(\theta_1(t) - \theta_2(t)) \\
 & + (m_1 + m_2) g l_1 \cos(\theta_1(t)) + m_2 g l_2 \cos(\theta_2(t)).
 \end{aligned} \tag{4.8}$$

Based on this Lagrangian *Mathematica* can create the Euler–Lagrange differential equations<sup>1</sup> and numerically solve the system for  $\theta_1$  and  $\theta_2$ . We can then translate the positions of each mass into a Cartesian plot and trace the trajectories. In brief, the *Mathematica* code looks like

```

<< VariationalMethods'

eqns = Eulerequations[L, {theta1[t], theta2[t]}, {t}];
eqnssolve = eqns /. {g -> 9.8, m1 -> 1, m2 -> 1, l1 -> 1, l2 -> 1}

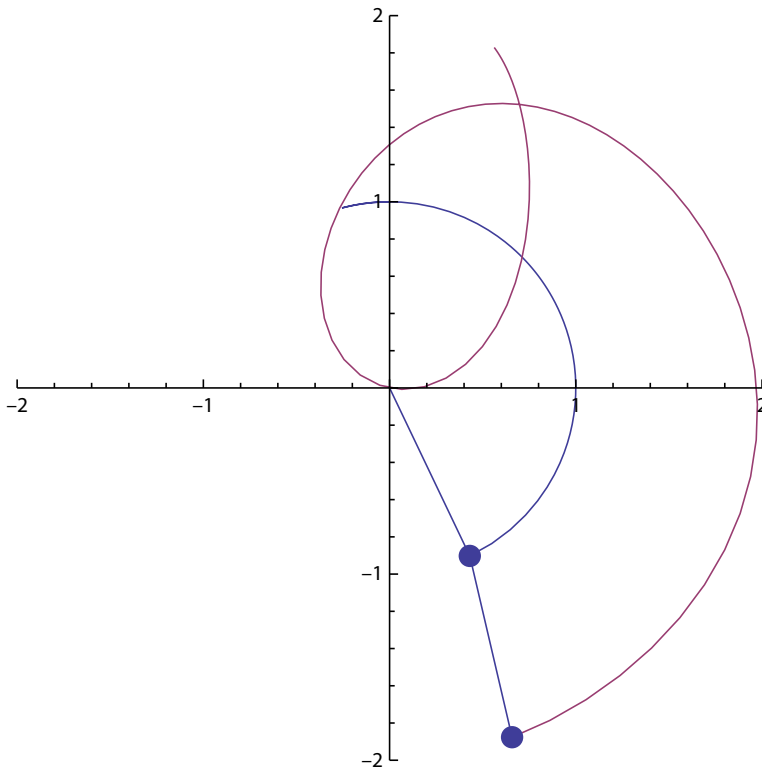
tstart = 0; (*Start time frame*)
tend = 20; (*Ending time frame for \
Manipulate*)
tstep = 0.025; (*Time step *)

```

The *VariationalMethod* import allows us to obtain the equations of motions by passing the Lagrangian to the function, and then solving for  $\theta_1$  and  $\theta_2$  as a function of time  $t$ .

We then use **NDSolve** to solve for  $\theta_1$  and  $\theta_2$ .

<sup>1</sup><http://reference.wolfram.com/language/VariationalMethods/ref/Eulerequations.html>



**Figure 4.3.** Motion of the double pendulum with two equal masses and massless rods of the same length.

```
sol = NDSolve[{eqnssolve[[1]], eqnssolve[[2]],
  theta1[0] == Pi, theta2[0] == Pi - 0.6,
  theta1'[0] == 0, theta2'[0] == 0
},
{theta1, theta2, {t, tstart, tend}
];
```

Figure 4.3 shows a plot tracing the path of the double pendulum solved via **NDSolve**. The reader should download the **Wolfram Demonstration** from the book website with the code to see the animation of the result.

### Exercises

1. Make plots varying the length of the pendulum. Look up the gravitational constants on other planetary bodies and vary that parameter as well, keeping SI units.
2. The original simple pendulum exercise assumes no damping factor. Add in a parameter to the differential equation that is proportional to the angular velocity and plot the solution you find with the function **NDSolve**.

## 4.3 Hydrostatics

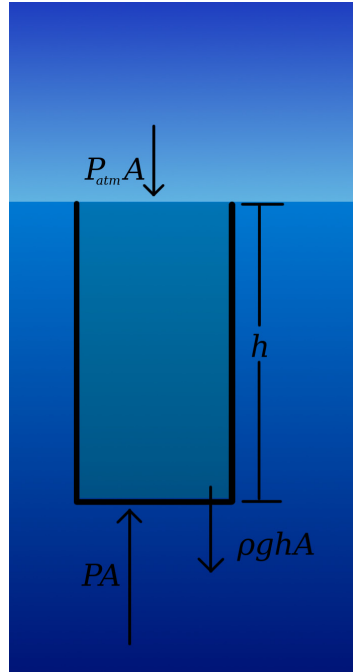
### Goals

- Use a simple example to understand how to illustrate Wolfram Alpha in *Mathematica*.
- Understand physical unit systems and Quantity function in *Mathematica*.
- Learn about pressure and density as it related to basic hydrostatics.
- Use free form language to obtain the appropriate physical quantities for the computation.
- Understand pressure differences within a water column.
- Determine the pressure at depth.

### Algorithm and code description

Atmospheric pressure at sea level is taken to be 14.696 psi or 101 325 Pascals at a temperature of 20 degrees Celsius [6]. This value, which we will call  $P_{\text{atm}}$  is the starting measurement from which we can compute the hydrostatic pressure surrounding a scuba diver underwater.

If we consider a cylindrical column of water with cross sectional area  $A$  and depth  $h$ , then the difference in forces at different pressures of the column must be equal to the weight of the water column. Thus the upward force below must be



**Figure 4.4.** A water column of depth  $h$  experiences a difference in force that is balanced by the weight of the water in a column of cross sectional area  $A$ .

greater than the force from atmospheric pressure above (figure 4.4). The force difference is given by

$$\Delta F = F_{\text{cyl}} \quad (4.9)$$

$$P_{\text{depth}} \times \text{Area} - P_{\text{atm}} \times \text{Area} = \rho gh \times \text{Area} \quad (4.10)$$

$$P_{\text{depth}} = P_{\text{atm}} + \rho gh. \quad (4.11)$$

We can take this opportunity to obtain physical data with Wolfram Alpha. This online service allows a user to do computations by entering natural language input and free-form linguistics. This can be employed in any *Mathematica* notebook by using the **equals (=)** prefix. If we want the density of water in SI units, simply input the following into a notebook

```
= density of water in kg per cubic meter. <Shift-Enter>
```

Furthermore, we can use the **Quantity** function in *Mathematica* to give physical units to our calculations.

```
Quantity[1000.0, "Kilograms"/"Meters"^3] *
  Quantity[9.8, "Meters"/"Seconds"^2] *
  UnitConvert[Quantity[34, "Feet"], "Meters"]

= 101559.kg m^{-2}
```

The density of fresh water is  $1.00 \times 10^3 \text{ kg m}^{-3}$  at standard atmospheric pressure and temperature. The average density of salt water in Earth's oceans is slightly higher at  $1.025 \times 10^3 \text{ kg m}^{-3}$ .

We can further convert this to convenient units of atmospheres (atm) to further demonstrate the power of the *Mathematica* notebook.

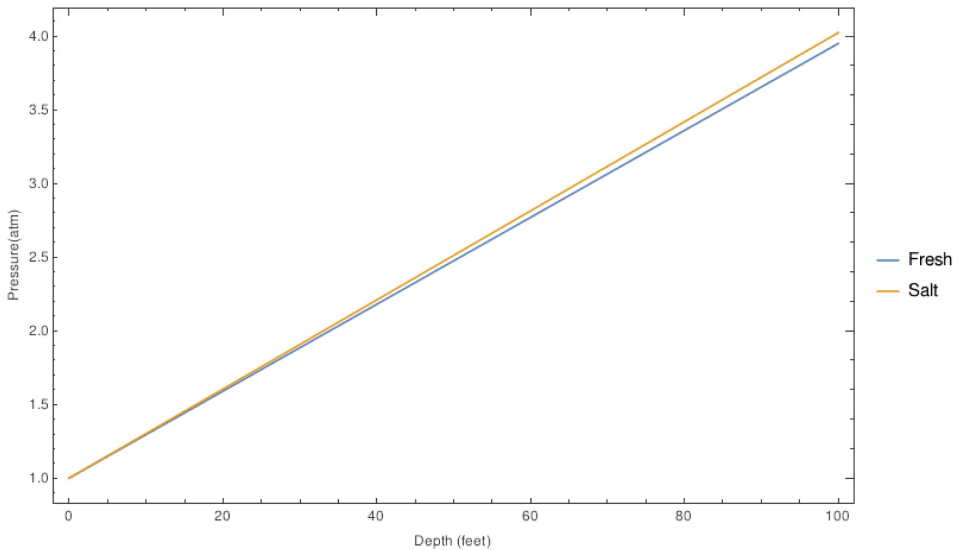
```
UnitConvert [
  Quantity[101559.,
    "Kilograms"/("Meters" "Seconds"^2)], "Atmospheres"]

= 1.00231 atm
```

Thus for every 34 feet a diver descends, the pressure increases by approximately 1 atmosphere. A comparison of the small, but important pressure difference at depth can be made via the following code (figure 4.5).

```
g = Quantity[9.8, "Meters"/"Seconds"^2];
rhofresh = Quantity[1000.0, "Kilograms"/"Meters"^3] ;
rhosalt = Quantity[1025.0, "Kilograms"/"Meters"^3] ;
```





**Figure 4.5.** Increasing pressure in atmospheres as a function of depth in feet for both fresh and salt water.

```
Plot[{Quantity[1.0, "Atmospheres"] + UnitConvert[(rhofresh * g
  * UnitConvert[Quantity[h, "Feet"], "Meters"]), "Atmospheres"],
  Quantity[1.0, "Atmospheres"] + UnitConvert[(rhosalt * g
  * UnitConvert[Quantity[h, "Feet"], "Meters"]), "Atmospheres"]
},
{h, 0, 100},
AxesLabel -> {"Depth (feet)", "Pressure(atm)"}]
```

### Exercises

1. Explore Wolfram Alpha to determine the density changes of water at different temperature for a constant pressure in the liquid phase. Remember to start with equals sign and keep the units consistent with **Quantity**.
2. The Mariana Trench has a depth of approximately 36 000 feet (~11 000 meters). What is the water pressure (salt water) in the column at this depth in psi? Assume the density of water is increased by approximately 5%.

## 4.4 Astronomical catalogs

### Goals

- Select a subset of the brightest stars in the sky.
- Use the Python `csv` module to import and filter an astronomical catalog.
- Create a polar grid of the northern celestial hemisphere.
- Use Matplotlib to scale the stellar magnitudes and plot the data.
- Indicate the  $0^\circ$  Galactic plane of the Milky Way Galaxy.

### Algorithm and code description

Astronomical catalogs have continued to increase in size as telescopes acquire more data at a higher rate, with more sensitive detectors, larger pixels and faster computing hardware. All-sky surveys like those conducted with the Very Large Array (VLA, [8]), Arecibo Radio Telescope [4], and the future Large Synoptic Survey Telescope (LSST, [7]) help us study stars, galaxies, time-domain events, and the large scale structure of the Universe. Catalogs are an excellent resource to utilize our Python skills on Raspberry Pi. For this example we will build a Python script to plot a catalog of the brightest stars in the night sky.

First, we define our prelude with module imports.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure, show, rc, grid
import csv
```

Next, we use the Python `csv.DictReader` module to import the formatted text data, skipping the first header line and only choosing stars in the northern celestial hemisphere and with apparent  $V$ -band magnitudes of  $V \leq 5.0$ . The data are listed in the General Catalogue of Photometric data<sup>2</sup> [3]. We specify as a Python list the names of all fields in the data file.

```
# get data
filename = 'bright.dat'
fields = ['_RAJ2000', '_DEJ2000', 'HR', 'Name',
          'HD', 'ADS', 'VarID', 'RAJ2000',
          'DEJ2000', 'Vmag', 'B-V', 'SpType', 'NoteFlag']
reader = csv.DictReader(open(filename), fields, delimiter='|')
# Skip first header line
next(reader)

# Define columns of interest
ra = []
dec = []
vmag = []

# Add in vertex elements with XY coordinates at each row
for row in reader:
    try:
        if ((float(row['_DEJ2000']) > 0.00) and
            (float(row['Vmag']) <= 5.0)):
            ra.append(float(row['_RAJ2000']))
            dec.append(float(row['_DEJ2000']))
            vmag.append(float(row['Vmag']))
    except:
        pass
```

<sup>2</sup><http://cdsarc.u-strasbg.fr/viz-bin/Cat?II/128>

Next we lay out a polar projection grid for declinations of  $0^\circ < \delta < +90^\circ$  and scale the points by  $V$ -band magnitude.

```
# Grid radar green, dotted lines
rc('grid', color='#006363', linewidth=1, linestyle=':')
rc('xtick', labelsizes=15, color='black')
rc('ytick', labelsizes=15, color='black')
rc('axes', edgecolor='white')

# make a square figure
size = 10
fig = figure(figsize=(size, size), facecolor='white')
ax = fig.add_axes([0.1, 0.1, 0.8, 0.8], polar=True, axisbg='black')

magscale = {'5':5, '4':10, '3':30, '2':40, '1':50, '0':100, '-1':100}

for i in range(len(ra)):
    ax.scatter(np.array(ra[i])*np.pi/180.0, 90.0-np.array(dec[i]),
              s=magscale[str(int(round(vmag[i])))],
              color='blue', edgecolor='white')
```

Finally we can use SkyCoord from the AstroPy module and draw the Galactic plane as a dashed line, and label the plot:

```
# Galactic Plane
import astropy.units as u
from astropy.coordinates import SkyCoord

gl = list(np.linspace(0, 360, 100))
gb = list(np.zeros(100))
gc = SkyCoord(l=gl*u.degree, b=gb*u.degree, frame='galactic')

raline=gc.fk5.ra.value
decline=gc.fk5.dec.value
raplot=[]
decplot=[]
for i in range(len(raline)):
    if decline[i] > 0.0:
        raplot.append(raline[i])
        decplot.append(decline[i])

ax.plot(np.array(raplot)*np.pi/180.0,
        90.0-np.array(decplot),
        lw=2, linestyle='-', color='#ff7400')

ax.set_rmax(90.0)
grid(True, color='grey')

ax.set_theta_direction(-1)
```

```

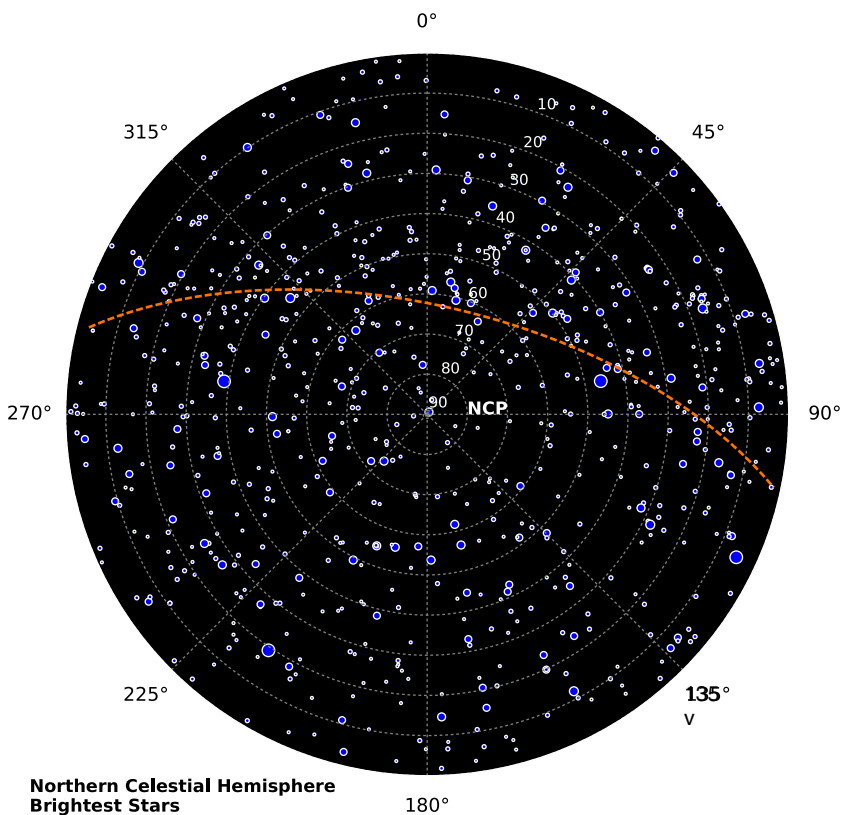
ax.set_theta_offset(np.pi/2)
ax.set_rgrids([1, 10, 20, 30, 40, 50, 60, 70, 80, 90],
              ['90','80','70','60','50','40','30','20','10','0'],
              20, fontsize=12, color='white')
ax.xaxis.label.set_color('white')
ax.yaxis.label.set_color('white')

ax.text(np.pi/2,10,'NCP',fontsize=14,fontweight='bold', color='white')
ax.text(225.0*np.pi/180.0,140,
        'Northern Celestial Hemisphere \nBrightest Stars',
        fontsize=14,fontweight='bold', color='black')
plt.savefig('polarstars.png')

plt.show()

```

The result, where you can see many patterns of stars known as asterisms within the constellations, is shown in figure 4.6.



**Figure 4.6.** Plot showing the brightest stars in the northern celestial hemisphere. The size of the points are scaled with apparent magnitude of the star. The dashed orange line indicates the Galactic plane of the Milky Way.

## Exercises

1. Download the NRAO VLA Sky Survey 1400 MHz radio survey [1] and use the same technique outlined in this section to plot the brightest radio sources in the northern sky. An example and data are provided on the book website.

## 4.5 The Lane–Emden equation

### Goals

- Understand a basic polytropic model of a star.
- Input the differential equation using the *Math Assist* interface.
- Numerically solve the Lane–Emden equation and tabulate the results for each polytropic index.
- Plot the results.

### Algorithm and code description

For stars that are in hydrostatic equilibrium, we assume a relation between pressure  $P$ , gravitational potential  $\Phi$ , and density  $\rho$  of

$$\frac{dP}{dr} = -\frac{d\Phi}{dr} \rho. \quad (4.12)$$

If a simple relation exists where the density does not depend on the temperature, then we can write a *polytropic* relation. Kippenhahn *et al* [5] gives a thorough overview of the background physics and derivation. We will examine the numerical solutions of the Lane–Emden equation, given by

$$\theta''(\xi) + 2\xi^{-1}\theta'(\xi) + \theta(\xi)^n = 0 \quad (4.13)$$

where dimensionless variables  $\theta$  and  $\xi$  relate to the density  $\rho$  and radius  $r$ . The equation is used in the study of polytropic gaseous spheres and the modeling of stars. Solutions of this second-order differential equation use polytropes to relate the pressure  $P$  and density  $\rho$  as a function of  $r$ , the radial coordinate measure from the center of the polytropic sphere (the star);  $P$  and  $\rho$  have the polytropic relation of

$$P = K\rho^{1+1/n} \quad (4.14)$$

where  $K$  is the polytropic constant and  $n$  is the polytropic index that we will manipulate in the *Mathematica* notebook.

The equation is solved with the polytropic index value  $n = 0, \dots, 5$  for the dimensionless function  $\theta(\xi)$ . We want to display the first root of each solution, derivatives of that root value, and the critical and mean density ratio. This will involve solving the differential equation in *Mathematica*. We group terms and rewrite the Lane–Emden equation as

$$\frac{1}{\xi^2} \frac{d}{d\xi} \left( \xi^2 \frac{d\theta}{d\xi} \right) + \theta^n = 0. \quad (4.15)$$

```
LaneEmden =  $\frac{1}{\xi^2} \partial_{\xi} (\xi^2 \partial_{\xi} \theta[\xi]) + \theta[\xi]^n == 0;$ 
sols = Table[NDSolve[ $\{$ LaneEmden /. n  $\rightarrow$  i,  $\theta$ [0.01] == 1,  $\theta'$ [0.01] == 0 $\}$ ,  $\theta$ ,  $\{\xi, 0.01, 10\}$ ,
MaxSteps  $\rightarrow$  100000], {i, 0, 5, 1}];
```

Figure 4.7. Lane–Emden equation syntax and method of solving the equation in *Mathematica*. The differential equation is solved with all exponents replaced with 0 through 5, evaluated with initial conditions (effectively at  $\xi = 0$ ) for  $\theta$  and  $\theta'$ .

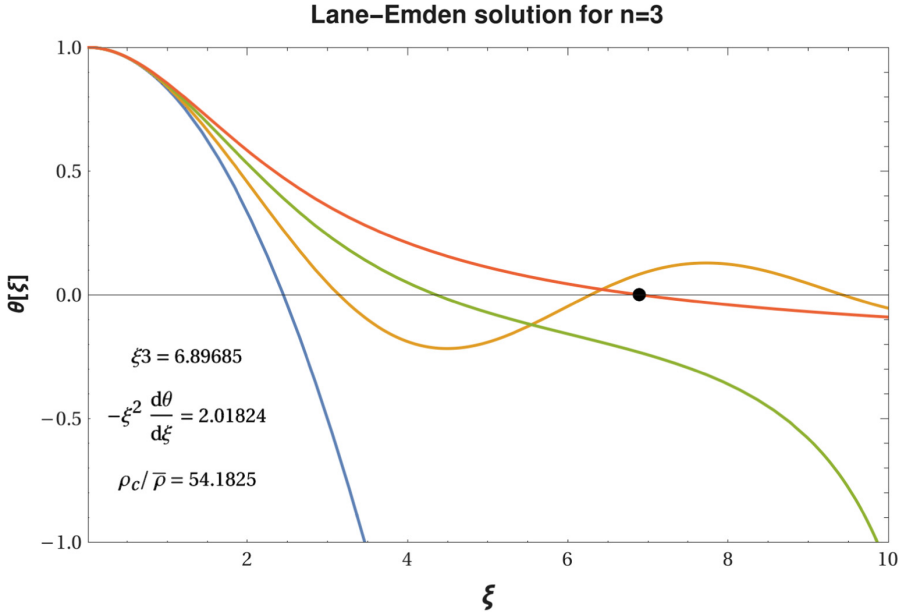


Figure 4.8. Example Lane–Emden solution with the root highlighted for  $n = 3$ .

This equation is shown typed directly into a *Mathematica* notebook in figure 4.7. Using the **Math Assist** typesetting commands we can input the symbols directly without any cumbersome syntax.

We numerically solve the equation with **NDSolve** for integer values of  $n$  and initial  $\theta$  and  $\theta'$  values of 1 and 0 respectively. Variable  $\xi$  goes from 0.01 to 10. The lower limit of  $\xi$  is such that a positive value to get convergence in the solver (figure 4.8).

A plot of the solution can be created with

```
nvalue = 3;
Plot[Theta[Xi] /. sols[[nvalue + 1]], {Xi, 0.01, 10}]
```

**Exercises**

1. Compute the roots of the solutions for  $n = 1, 2, 3, 4$  and 5. Use the Wolfram Demonstrations CDF file to aid in your determination located at <http://demonstrations.wolfram.com/LaneEmdenequationInStellarStructure/>.

## 4.6 Radiative transfer

### Goals

- Understand the equation and parameters of radiative transfer.
- List all parameters associated with the general differential equation.
- Understand the different regimes of optical depth.

### Algorithm and code description

The equation of radiative transfer describes the propagation of radiation and the effects of emission, absorption, and scattering through a medium. *Mathematica* can solve for the intensity  $I_\nu$  as a function of optical depth. This section shows the simple case of an initial intensity through a volume of gas with no scattering, constant opacity, gas density, and source function intensity.

The equation of radiative transfer is given by

$$\frac{1}{\rho} \frac{dI_\nu}{ds} = -\kappa_\nu I_\nu + j_\nu \quad (4.16)$$

where  $I_\nu$  is the specific intensity,  $\rho$  is the gas density,  $\kappa_\nu$  is the opacity or absorption coefficient, and  $j_\nu$  is the emission coefficient. The equation describes how incident radiation is affected along a path length  $s$ . We define the source function  $S_\nu$  as well as the optical depth  $\tau_\nu$

$$S_\nu = \frac{j_\nu}{\kappa_\nu} \quad (4.17)$$

$$\tau_\nu = \int_0^s \kappa_\nu \rho \, ds \quad (4.18)$$

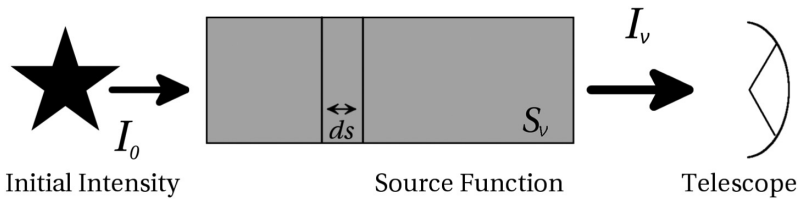
and can rewrite the equation of radiative transfer in terms of  $\tau$

$$\frac{dI_\nu}{d\tau} = -I_\nu + S_\nu. \quad (4.19)$$

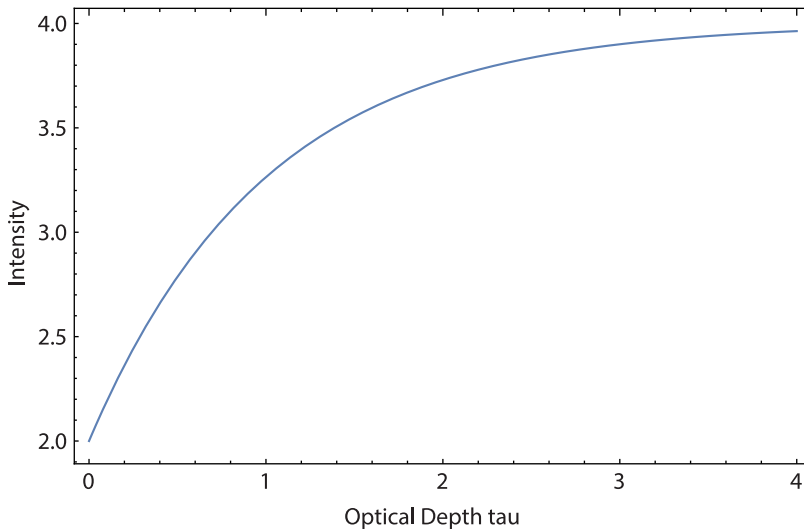
The formal solution for the specific intensity at a given frequency  $\nu$  for a zero angle of incidence (plane-parallel) is

$$I_\nu(\tau) = I_0 e^{-\tau} + \int_0^\tau S_\nu(t) e^{-t} dt. \quad (4.20)$$

Figure 4.9 shows a schematic diagram of this phenomenon. The intensity approaches the source function intensity for optically thick cases ( $\tau \gg 1$ ) and the initial intensity of the background source  $I_0$  for optically thin cases ( $\tau \ll 1$ ).



**Figure 4.9.** Radiative transfer diagram. The initial intensity is exponentially attenuated by the intervening ‘cloud’ and source function along the line of sight to the telescope. The source function also contributes to the emission and is integrated along the path  $s$ . The final intensity  $I_v$  is what is observed as the sum of these components.



**Figure 4.10.** Solution of the radiative intensity  $I_v$  versus optical depth  $\tau$ .

The following *Mathematica* code allows us to directly input the differential equation and solve it.

```
eq1 = D[inu[tau], tau] == -inu[tau] + S0;
sol = DSolve[{eq1, inu[0] == i0}, inu[tau], tau];
```

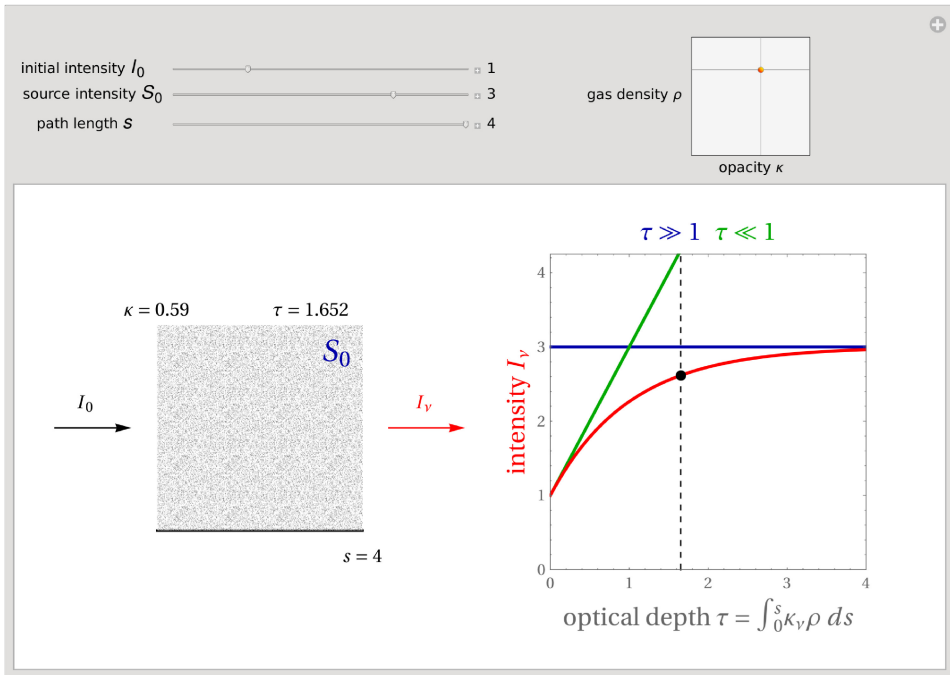
As an example we can plot  $I_v$  versus  $\tau$  for arbitrary values of  $I_0$  and  $S_0$  (figure 4.10).

Figure 4.11 shows a GUI that the reader can download to study the code and vary the parameters discussed in this section.

**Exercises**

1. Write code for and plot the specific intensity for optically thin cases ( $\tau \ll 1$ ).
2. Use the Wolfram Demonstrations CDF file to vary the density  $\rho$ , opacity  $\kappa$ , and path length  $s$  to study the effects on the specific intensity  $I$  at <http://demonstrations.wolfram.com/ComputationOfRadiativeTransfer/>.





**Figure 4.11.** Wolfram Demonstrations app for radiative transfer. The three sliders control the initial intensity, source intensity, and path length. The orange joystick changes the gas density and opacity.

## References

- [1] Condon J J, Cotton W D, Greisen E W, Yin Q F, Perley R A, Taylor G B and Broderick J J 1998 The NRAO VLA Sky Survey *Astron. J.* **115** 1693–716
- [2] Goldstein H, Poole C P and Safko J L 2002 *Classical Mechanics* (Reading, MA: Addison Wesley)
- [3] Hauck B, Nitschelm C, Mermilliod M and Mermilliod J-C 1990 The general catalogue of photometric data *Astronomy & Astrophysics Suppl.* **85** 989–98
- [4] Kent B R, Giovanelli R, Haynes M P, Martin A M, Saintonge A, Stierwalt S, Balonek T J, Brosch N and Koopmann R A 2008 The Arecibo Legacy Fast Alfa survey. VI. Second HI source catalog of the Virgo cluster region *Astron. J.* **136** 713–24
- [5] Kippenhahn R, Weigert A and Weiss A 2012 *Stellar Structure and Evolution*. (Berlin: Springer)
- [6] Linstrom P J and Mallard W G 2001 The nist chemistry webbook: A chemical data resource on the internet *Journal of Chemical & Engineering Data* **46** 1059–63
- [7] LSST Science Collaboration; Abell P A, Allison J, Anderson S F, Andrew J R, Angel J R P, Armus L, Arnett D, Asztalos S J and Axelrod T S *et al* 2009 *LSST Science Book* (Version 2.0) [e-prints](#)
- [8] Perley R A, Chandler C J, Butler B J and Wrobel J M 2011 The expanded very large array: a new telescope for new science *ApJ Letters* **739**–L1

# Chapter 5

## Machine learning

### 5.1 Spanning trees

#### Goals

- Install a new Python module.
- Understand how a minimum spanning tree is constructed.
- Plot the results and use the trees to identify clusters.

#### Algorithm and code description

Spanning trees represent a graph of connected points with edges that can be assigned weights. A Minimum Spanning Tree (MST) can be generated that minimizes the computed sum of weights and therefore a network of connecting edges, with no closed loops in the graph [2, 3]. Figure 5.1 shows the construction of the MST with purple lines. The generation of these graphs for a collection of points represents an unsupervised machine learning technique [5].

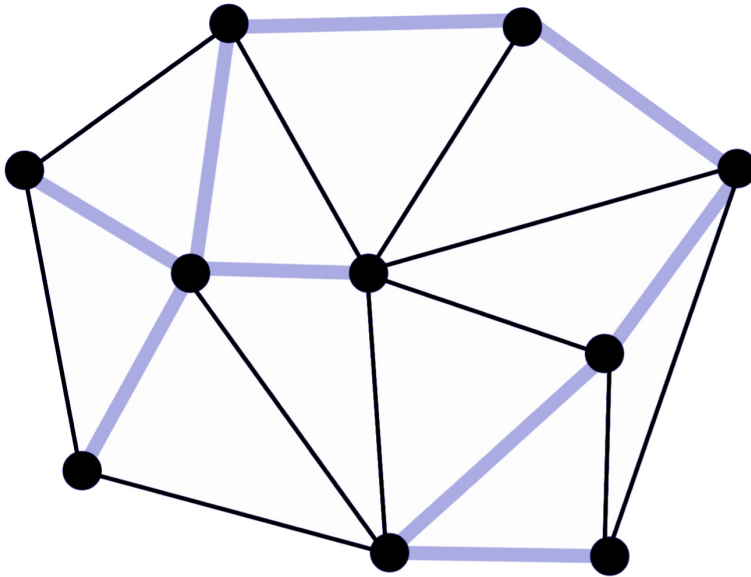
Using the *mst\_clustering* Python module with *scikit-learn*, we can create and examine MSTs [9]. The required modules are installed with the commands

```
conda install numpy scipy scikit-learn
conda install pip
pip install mst_clustering
```

If downloading the module stand alone

```
python setup.py install
```

Now, entering an IPython prompt, we can create a pseudo random set of  $x$  and  $y$  values and create an MST.



**Figure 5.1.** Minimum Spanning Tree (MST) where the tree is shown by the purple lines. The thin black lines show all possible connections.

```
import math
import numpy as np
from random import random
from mst_clustering import MSTClustering

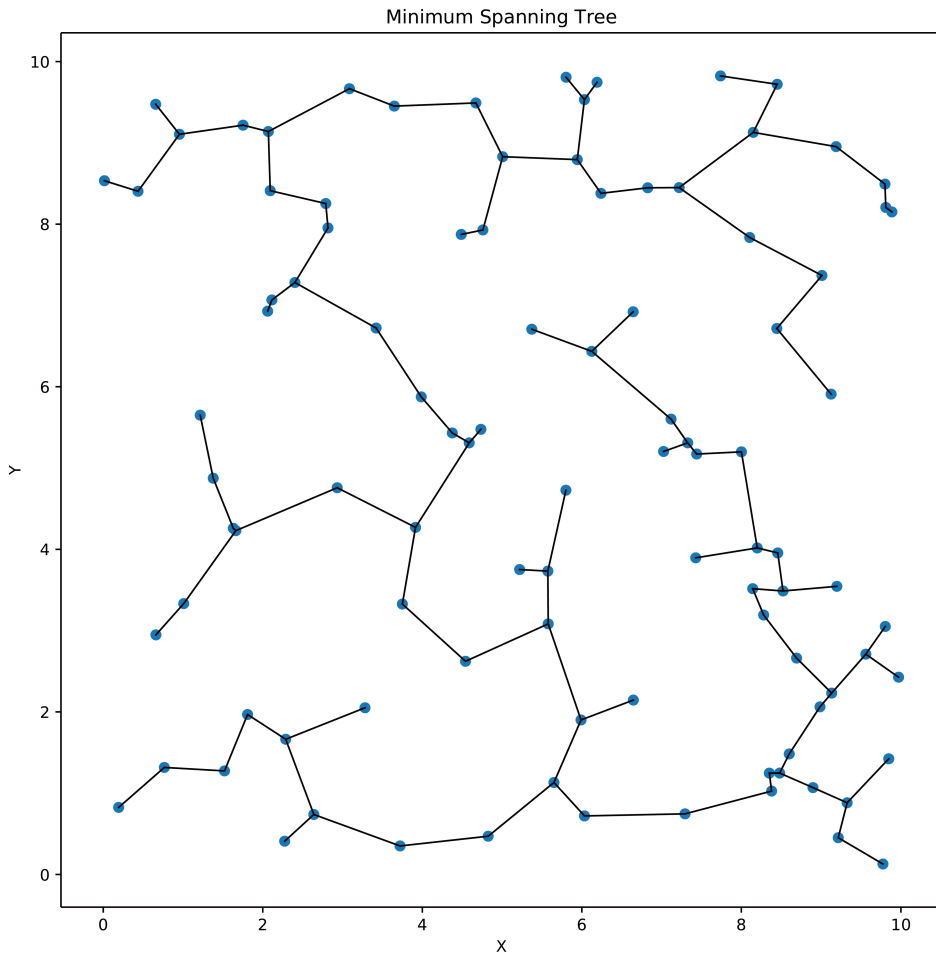
# Random points
minimum = 0
maximum = 10

cluster = []
for i in range(100):
    x = minimum + (maximum - minimum) * random()
    y = minimum + (maximum - minimum) * random()
    cluster.append([x,y])
model = MSTClustering(cutoff_scale=2, approximate=False)
labels = model.fit_predict(cluster)
```

The **model** Python object and associated class methods will group points together and create the MST. We can then plot the points and their associated connections (figure 5.2).

```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10,10), facecolor='white')
ax = fig.add_subplot(111)
```



**Figure 5.2.** Minimum Spanning Tree example and resulting connections drawn between a pseudo randomly generated sample of points.

```
X = model.X_fit_
segments = model.get_graph_segments(full_graph=True)

ax.plot(segments[0], segments[1], '-k', zorder=1, lw=1)
ax.scatter(X[:, 0], X[:, 1])
plt.show()
```

We can now expand into a slightly more complex data input and MST exercise using the positions of galaxies in clusters. Galaxy group classification has been carried out in a number of ways. We use the group classification published as the Abell catalog of rich clusters [1]. Building on the examples and data from Dressler

and Rhee *et al* [4, 7], we can import formatted text data into a list of Python dictionaries. We use the same `csv` reader class to import an astronomical catalog into a list of Python dictionaries.

```
import collections
import csv
import numpy as np
import matplotlib.pyplot as plt
from mst_clustering import MSTClustering

filename = 'dressler2.tsv'

fields=['RAJ2000', 'DEJ2000', 'Cluster', 'Gal', 'Xpos', 'RA1950',
        'DE1950', 'MType', 'mvis', 'mbulge', 'ell', 'cz', 'n_cz',
        '_RA.icrs', '_DE.icrs']

reader = csv.DictReader(open(filename), fields, delimiter='|')

dicts = []

#Skip three header lines
next(reader)
next(reader)
next(reader)
for row in reader:
    dicts.append(row)
```

The next section of code sorts the list of dictionaries into a Python collection.

```
# Sort galaxies into cluster groups
clusternames=[]
for row in dicts:
    clusternames.append(row['Cluster'])

clusterset = set(clusternames)
clusternames = list(clusterset)

catalog = collections.defaultdict(list)

for cluster in clusternames:
    for row in dicts:
        if cluster in row['Cluster']: catalog[cluster].append(row)
```

Finally, we will compute the mean Right Ascension and Declination ( $x$  and  $y$  in this example) to determine the relative positions of each cluster member. The MST is

computed, and graphed in a grid of 4 x 3 plots with Matplotlib. We also modify the plot axis labels to be more of a publication quality type text (figure 5.3).

```

fig = plt.figure(figsize=(12,16), facecolor='white')

count=0
for clusterselect in catalog.keys():

    print(clusterselect)
    xcoords = []
    ycoords = []

    for row in catalog[clusterselect]:
        xcoords.append(float(row['RAJ2000']))
        ycoords.append(float(row['DEJ2000']))

    xcentroid = np.mean(xcoords)
    ycentroid = np.mean(ycoords)

    Y=[]

    for row in catalog[clusterselect]:
        Y.append([float(row['RAJ2000'])-float(xcentroid),
                 float(row['DEJ2000'])-float(ycentroid)])

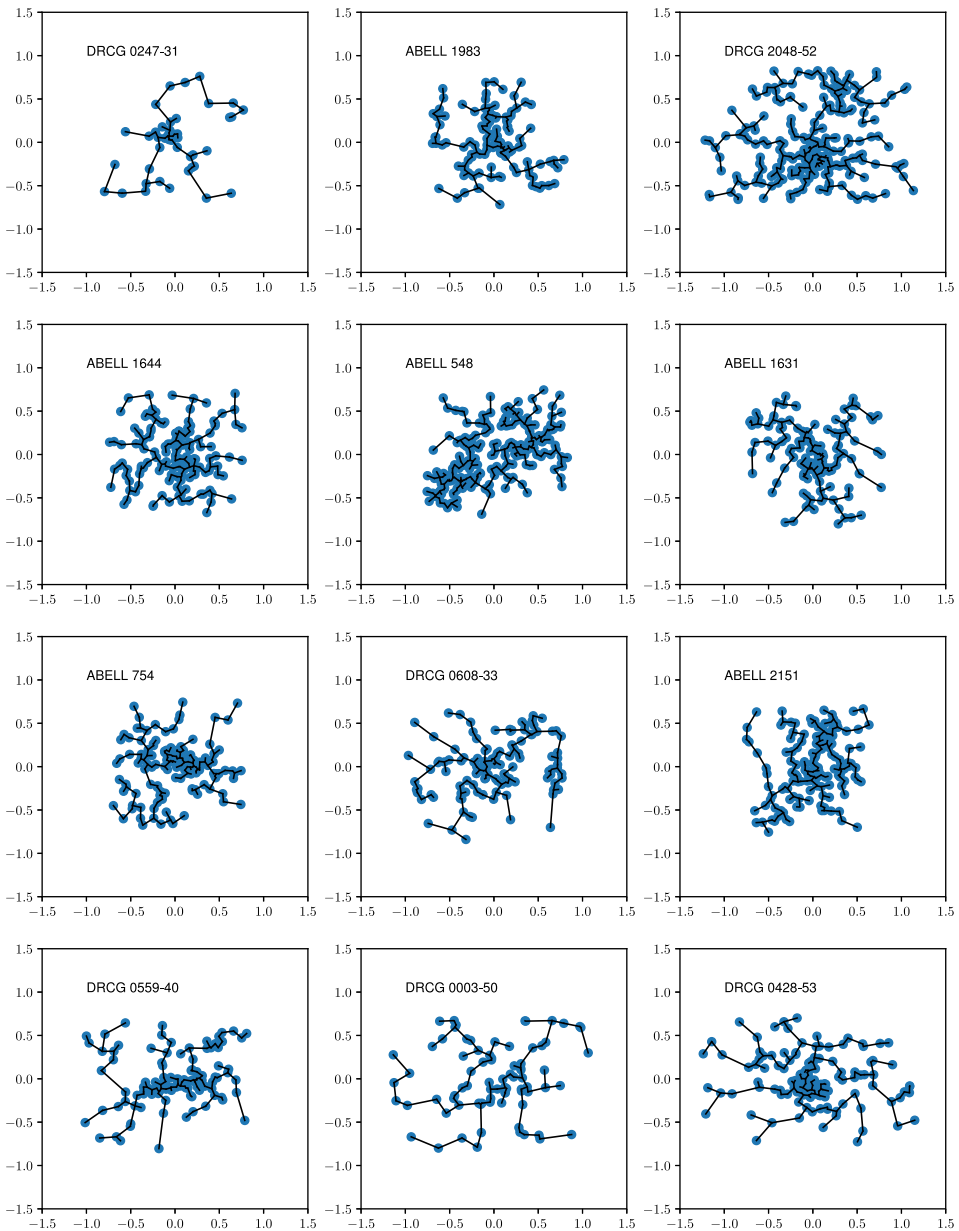
    model = MSTClustering(cutoff_scale=2, approximate=False)
    labels = model.fit_predict(Y)
    X = model.X_fit_
    segments = model.get_graph_segments(full_graph=True)
    ax = plt.subplot(4,3, count+1)
    ax.plot(segments[0],segments[1], '-k', zorder=1, lw=1)
    ax.scatter(X[:, 0], X[:, 1])

    ax.set_xlim(-1.5,1.5)
    ax.set_ylim(-1.5,1.5)
    ax.text(-1, 1, clusterselect)

    count = count + 1

plt.rc('text', usetex=True)
plt.rc('font', **{'family':'sans-serif','sans-serif':['Helvetica']})
plt.draw()
plt.savefig('abell.png')
plt.show()

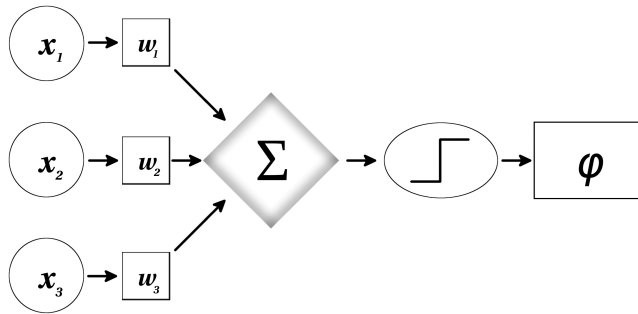
```



**Figure 5.3.** Minimum Spanning Trees of 12 Abell Clusters. The  $x$ - and  $y$ -axis show the relative Right Ascension and Declination positions of galaxies within each cluster.

### Exercises

1. The example given in the `mst_clustering` documentation shows how clusters can be identified with a ‘trimmed’ minimum spanning tree. Compare the results of an MST with those of the `sklearn.cluster` module.



**Figure 5.4.** Neural network where input nodes are weighted, summed, and then passed through an activation function to an output node.

2. Compare the results of the `mst_cluster` Python module to the `Mathematica` function `FindSpanningTree`<sup>1</sup>.

## 5.2 Neural networks and classification

### Goals

- Understand the basics of an artificial neural network system.
- Understand how to obtain and define a training set.
- Import a small training dataset.
- Use the function `Classify[]` to study the training data.
- Compare two different *Mathematica* classification functions and their probability calculations.
- Use `ClassifyCluster` similar to previous machine learning exercises.

### Algorithm and code description

Artificial neural networks mimic the same pathways and connections observed in neurological systems [8]. Each artificial neuron in the system acts as a node just like in the brain. Each node has inputs that are weighted, summed and then given a binary status (ON/OFF—a Heaviside step function) in the simplest form of an activation function (figure 5.4).

The array of outputs is subject directly to the inputs and pre-programmed function. Mathematically,


$$\phi(\mathbf{x} \cdot \mathbf{w}) = \phi\left(\sum_{i=1}^n (x_i \cdot w_i)\right) \quad (5.1)$$

where  $\mathbf{x}$  is the input vector and  $\mathbf{w}$  represents the weighting function. There can be multiple layers of inputs nodes that filter into a single output element. A simple yet effective function in *Mathematica* is `Classify[]`. We can pass it a training data set and use the resulting object to categorize an image.

<sup>1</sup><https://reference.wolfram.com/language/ref/FindSpanningTree.html>




```
In[24]:= obj = ResourceObject["CIFAR-10"];
trainingList = ResourceData[obj, "TrainingData"];
trainingDataSubset = RandomSample[trainingList, 500];
classifierObject = Classify[trainingDataSubset];
```

```
In[28]:= classifierObject[
```

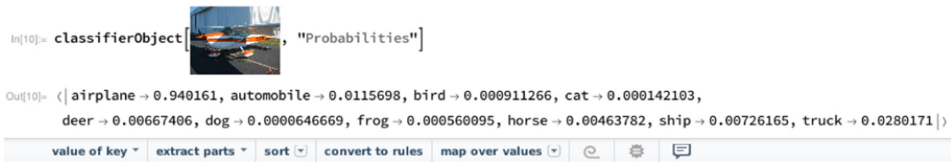
Out[28]= airplane



**Figure 5.5.** `Classify` is used on the CIFAR-10 training data. *Mathematica* allows the user to directly paste an image into the resulting classifier Object and give a result.

```
In[10]:= classifierObject[, "Probabilities"]
```

Out[10]= { | airplane → 0.940161, automobile → 0.0115698, bird → 0.000911266, cat → 0.000142103,  
deer → 0.00667406, dog → 0.0000646669, frog → 0.000560095, horse → 0.00463782, ship → 0.00726165, truck → 0.0280171 | }



**Figure 5.6.** With keyword ‘Probabilities’ passed as a keyword argument, we can see that the input image is classified as an airplane with a certainty of 94%.

The CIFAR-10 dataset<sup>2</sup> is used in many *Mathematica* documentation examples, and we will do a similar exercise here [6]. There are 60 000 images in the data—for illustrating this example on a Raspberry Pi, we will choose a random sample of 500 images for our training set as it takes less time to execute, but still gives appreciable results. The commands are relatively straight forward:

```
obj = ResourceObject["CIFAR-10"];
trainingList = ResourceData[obj, "TrainingData"];
trainingDataSubset = RandomSample[trainingList, 500];
classifierObject = Classify[trainingDataSubset];
```

We can now pass any image to this `classifierObject`. *Mathematica*’s interface gives a user the ability to directly copy and paste an image directly into a function’s arguments via the notebook (figure 5.5).

We can also determine a probability distribution to understand the quantify the confidence in our calculation (figure 5.6).

We can complete a similar exercise to our minimum spanning tree in section 5.1 and use `ClusterClassify`. We specify *a priori* information into how many groups we wish to break down the clusters.

<sup>2</sup><https://www.cs.toronto.edu/kriz/cifar.html>

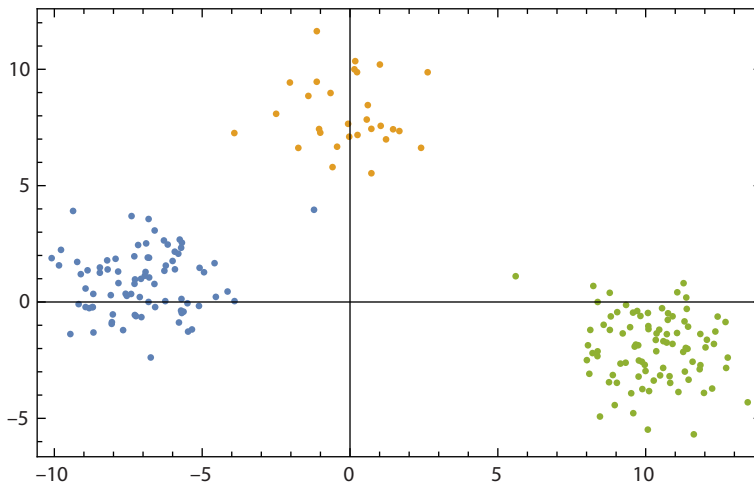


Figure 5.7. `ClusterClassify` can be used to group various distribution sets.

First, define cluster of pseudo random data<sup>3</sup>:

```
distribution = MixtureDistribution[{1, 1, 0.5}, {
  MultinormalDistribution[{10, -2}, {{2, 0}, {0, 2}}],
  MultinormalDistribution[{-7, 1}, {{2, 0}, {0, 2}}],
  MultinormalDistribution[{0, 8}, {{2, 0}, {0, 2}}]
}];
data = RandomVariate[distribution, 200];
ListPlot[data, PlotRange -> All]
```

Next, classify the data into clusters, letting the function decide how many cluster it will break things down into (obviously three)

```
cl = ClusterClassify[data]
```

We can use the Boolean selection Function **Pick** to select elements of a particular cluster identification and color code accordingly.

```
ListPlot[Pick[data, cl[data], #] & /@ {1, 2, 3}, Frame->True]
```

Figure 5.7 shows the color-coded result of this classification scheme.

## Exercises

1. Study the CIFAR-10 training data set and determine the classification probabilities with some of your own images. How well does the function determine different subclasses of objects?
2. Use `ClusterClassify` to separate out a sparse dataset where the distributions are not distributed so far apart.

<sup>3</sup><http://reference.wolfram.com/language/ref/ClusterClassify.html>

## References

- [1] Abell G O 1958 The Distribution of Rich Clusters of Galaxies. *Astrophys. J. Suppl.* **3** 211
- [2] Barrow J D, Bhavsar S P and Sonoda D H 1985 Minimal spanning trees, filaments and galaxy clustering *Mon. Not. R. Astron. Soc.* **216** 17–35
- [3] Cartwright A and Whitworth A P 2004 The statistical analysis of star clusters *Mon. Not. R. Astron. Soc.* **348** 589–98
- [4] Dressler A and Shectman S A 1988 Evidence for substructure in rich clusters of galaxies from radial-velocity measurements *Astron. J.* **95** 985–95
- [5] Ivezić Ž, Connelly A J, VanderPlas J T and Gray A 2014 *Statistics, Data Mining, and Machine Learning in Astronomy* (Princeton, NJ: Princeton University Press)
- [6] Krizhevsky A, Nair V and Hinton G 2009 *Cifar-10* (<http://www.cs.toronto.edu/~kriz/cifar.html>).
- [7] Rhee G F R N, van Haarlem M P and van Katgert P 1991 Substructure in Abell clusters *Astron. Astrophys.* **246** 301–12
- [8] Schmidhuber J 2015 Deep learning in neural networks: an overview *Neural Netw.* **61** 85–117
- [9] VanderPlas J 2016 mst clustering: clustering via Euclidean minimum spanning trees *J. Open Source Software* **1** 12

# Chapter 6

## Image combination and analysis

### 6.1 Image manipulation

#### Goals

- Understand online sources of planetary imaging.
- Install the Python Imaging Library (PIL).
- Learn about different image display options in Python.
- Create an RGB image from astronomical filter bands.
- Create a movie of an astronomical data cube.

#### Algorithm and code description

Installation of the Python Imaging Library<sup>1</sup> via the Pillow fork is completed via **pip**

```
pip install Pillow
```

The basic command of opening and showing an image are

```
from PIL import Image
im = Image.open("galaxy.png")
im.show()
```

We will use some of the latest scientific imaging from Juno. The NASA Juno spacecraft mission, launched in 2011, is currently orbiting Jupiter in a polar orbit [2, 4]. The instrumentation will study the magnetosphere and composition of Jupiter.

We can download a set of Jupiter images<sup>2</sup> and use the PIL to generate a red–green–blue (RGB) image. This example creates a function such that a Python list of

---

<sup>1</sup><https://pillow.readthedocs.io/>

<sup>2</sup><https://www.missionjuno.swri.edu/junocam/processing?id=4116>

the red, green, and blue file names is passed as a function keyword argument, and the image is combined and displayed.

```

from PIL import Image
import numpy as np

def combineImages(rgbfiler):

    RGB = []

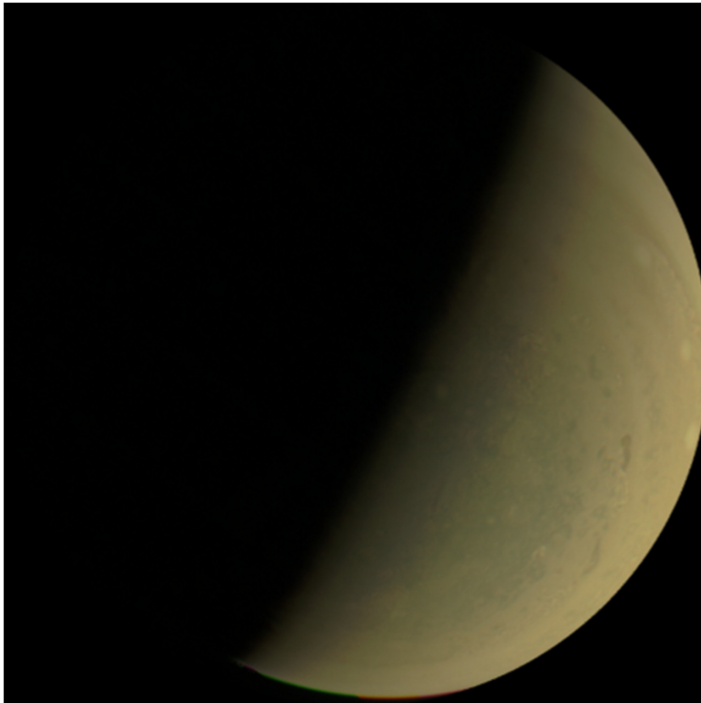
    for i, filename in enumerate(rgbfiler):
        im = Image.open(filename)
        implane = np.array(list(im.getdata()), dtype="uint8")
        reshape = implane.reshape(im.size[0], im.size[1])
        RGB.append(Image.fromarray(reshape, mode=None))

    combine = Image.merge("RGB", (RGB[0], RGB[1], RGB[2]))
    combine.show()

# Images must be in the order of red, green, blue
combineImages(['red.png', 'green.png', 'blue.png'])

```

Figure 6.1 shows the resulting combined image.



**Figure 6.1.** Juno mission multi-color image of the planet Jupiter.

## Exercises

1. Look at the JunoCam website<sup>3</sup> and examine images from different periapse (closest approach) segments of the mission. Combine those images using the function written in this section.

## 6.2 Creating a multi-wavelength astronomical image

### Goals

- Understand online sources of astronomical images and surveys.
- Create an RGB image from FITS files in different astronomical filter bands.

### Algorithm and code description

Studies of astronomical objects cover the entire range of the electromagnetic spectrum. In this example we will use images from three different parts of the EM spectrum to show that our Galaxy has different components only seen at certain wavelengths.

Astronomical images are typically made available in a scientific format called the *Flexible Image Transport System* (FITS, [7]).

The first example will use all sky surveys with data in the optical *B* blue band [9], red for *H $\alpha$*  [5], and green for low frequency 408 MHz radio [8] (figure 6.2).

We will use AstroPy to load the FITS files [1, 6].

```
from PIL import Image
import numpy as np
from astropy.io import fits

def combineImages(rgbafiles, output):

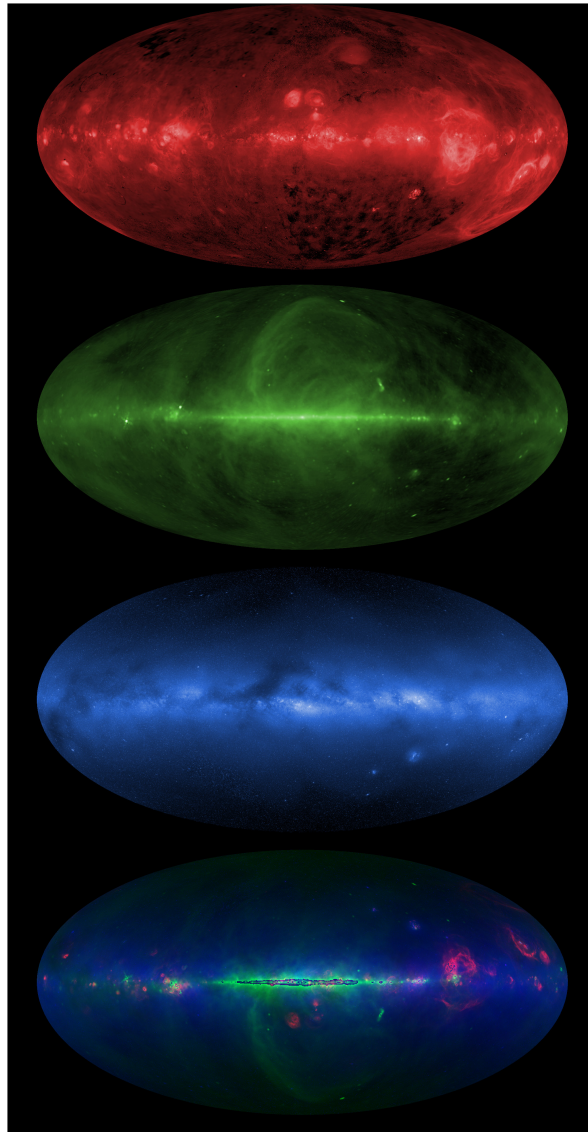
    RGBA = []

    for i, filename in enumerate(rgbafiles):
        hdul = fits.open(filename)
        hdu = hdul[0]
        implane = np.array(list(hdu.data), dtype="uint8")
        reshape = implane.reshape(hdu.shape[0], hdu.shape[1])
        RGBA.append(Image.fromarray(reshape, mode=None))
        hdul.close

    combine = Image.merge("RGBA", (RGBA[0], RGBA[1], RGBA[2]))
    combine.show()
    combine.save(output)

combineImages(['halpha.fits', '408.fits', 'mellingerblue.fits'],
              'milkyway.png')
```

<sup>3</sup><https://www.missionjuno.swri.edu/junocam/processing>



**Figure 6.2.** All-sky Hammer–Aitoff projections of the Milky Way Galaxy. The top red panel shows the sky through an  $H\alpha$  filter. The green panel shows the radio sky at a frequency of 408 MHz. The blue panel shows the optical sky, and the final bottom panel shows the images combined with the Python Imaging Library.

The second example uses data from the National Radio Astronomy Observatory (NRAO<sup>4</sup>) archive at C-band (4.7 GHz), the x-ray Telescope on the Swift spacecraft<sup>5</sup>, and the Wide-field Infrared Survey Explorer (WISE)<sup>6</sup> at 12 microns. The

<sup>4</sup> <https://www.nrao.edu/>

<sup>5</sup> <https://swift.gsfc.nasa.gov/>

<sup>6</sup> <https://www.nasa.gov/missionpages/WISE/main>

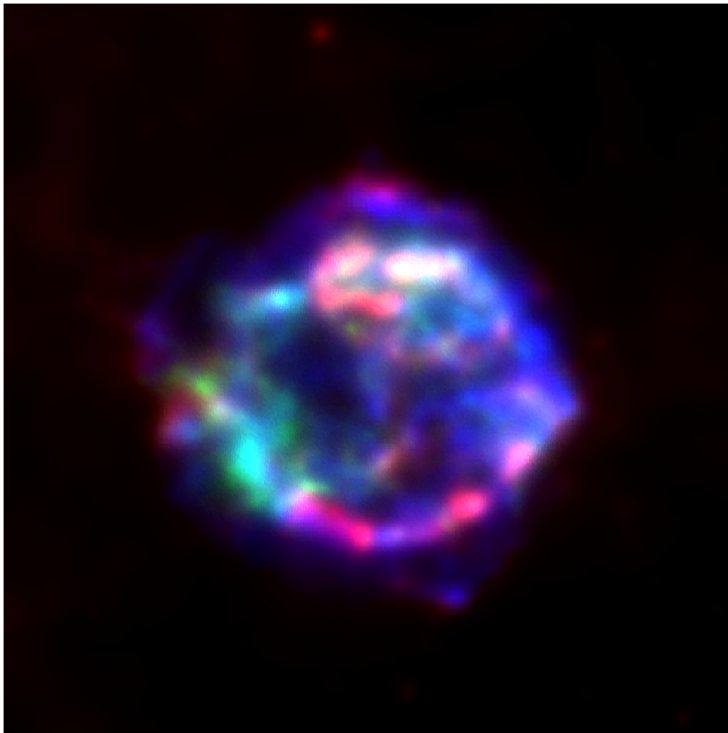
astronomical object to be imaged is a supernova remnant called *Cassiopeia A* [3]. The function for the Milky Way example can be reused.

```
combineImages(['wise12.fits', 'xray.fits', 'radio.fits'], 'cassA.png')
```

Figure 6.3 shows the result of combining the three images, revealing different high energy and physical processes that astronomers detect from a supernova remnant.

### Exercises

1. Use the SkyView service to download images of spiral Galaxy NGC 6946 in the near-ultraviolet, Digital Sky Survey 2 blue optical, and WISE 12-micron infrared. Use our *combineImages()* function to create an RGB image with the AstroPy Python module.



**Figure 6.3.** Supernova remnant Cassiopeia A, a combined color image of C-band radio from the Very Large Array, x-ray from the Swift spacecraft, and WISE infrared telescope at 12 microns.



## 6.3 Manipulating astronomical data cubes

### Goals

- Create a movie of an astronomical data cube using Matplotlib.

### Algorithm and code description

Matplotlib, numpy, pylab/Scipy, astropy, and APLpy are needed for this example. The example used below is a naturally weighted, VLA HI data cube of Galaxy NGC 2841 (figure 6.4), and can be obtained from this listing of HI data cubes<sup>7</sup> [11].



**Figure 6.4.** Spiral Galaxy NGC 2841—red is WISE 3.4 micron, green is Digital Sky Survey 2 blue plates, and blue is GALEX far ultraviolet.

---

<sup>7</sup><http://www.mpia.de/THINGS/Overview.html>

The module imports needed are

```
import os, sys, string, aplpy
import matplotlib.pyplot as plt
import numpy as np
import pylab as py
from astropy.io import fits
from matplotlib import colors, cm
```

We can define a custom colormap in matplotlib with

```
cmap = ((0.0, 0.0, 0.0), (0.25,0.0,0.0),
        (0.5, 1.0, 1.0), (0.75,1.0,1.0),
        (1.0, 1.0, 1.0))

cdict_customcmap = {
    'red' : cmap,
    'green': cmap,
    'blue' : cmap
}

customcmap = colors.LinearSegmentedColormap('customcmap',
                                             cdict_customcmap, 1024)
```

These parameters can be hardwired into a script, used as function keyword arguments, or prompted for at the terminal depending on how one wishes to interact with the data.

```
filename='NGC_2841_NA_CUBE_THINGS.FITS'
nchan=128
img=fits.getdata(filename)
hdulist = fits.open(filename) # open a FITS file
prihdr = hdulist[0].header # the primary HDU header

# FITS coordinate pixels
size = 1
ramin = 462 - size
ramax = 462 + size
decmin = 575 - size
decmax = 575 + size

# Select a spectrum out of the data cube
# Units of mJy/beam and km/s
# Creating the abscissa array will vary depending on the datacube
spectrum = [np.mean(img[i,decmin:decmax,
                    ramin:ramax])*1000.0 for i in range(0,nchan)]
velocity = [(prihdr['CRVAL3'] +
            prihdr['CDELTA3']*(i+prihdr['CRPIX3']))/1000.0
            for i in range(0,nchan)]
```

We now utilize *APLpy*<sup>8</sup> and Matplotlib to slice through a data cube channel along the frequency/velocity axis [10]. Note that a ‘recenter’ class method can be used to crop the field as desired. We use an `os.system` command to create a ‘*pngs*’ subdirectory if it does not already exist.

The data in the FITS cube are in a flux density unit of milliJanskys per beam, where 1 Jansky =  $10^{-26} \text{ W m}^{-2} \text{ Hz}^{-1}$ .

Figure 6.5 shows an example data cube image and spectrum.

```
fig = plt.figure(facecolor='w', edgecolor='w',
                 frameon=True, figsize=(6,7))

if not os.path.exists('pngs'):
    os.makedirs('pngs')

print('WARNING: Removing old *.png files from pngs directory...')
os.system('rm -rf pngs/*.png')

for i in range(0,nchan):
    plt.clf()

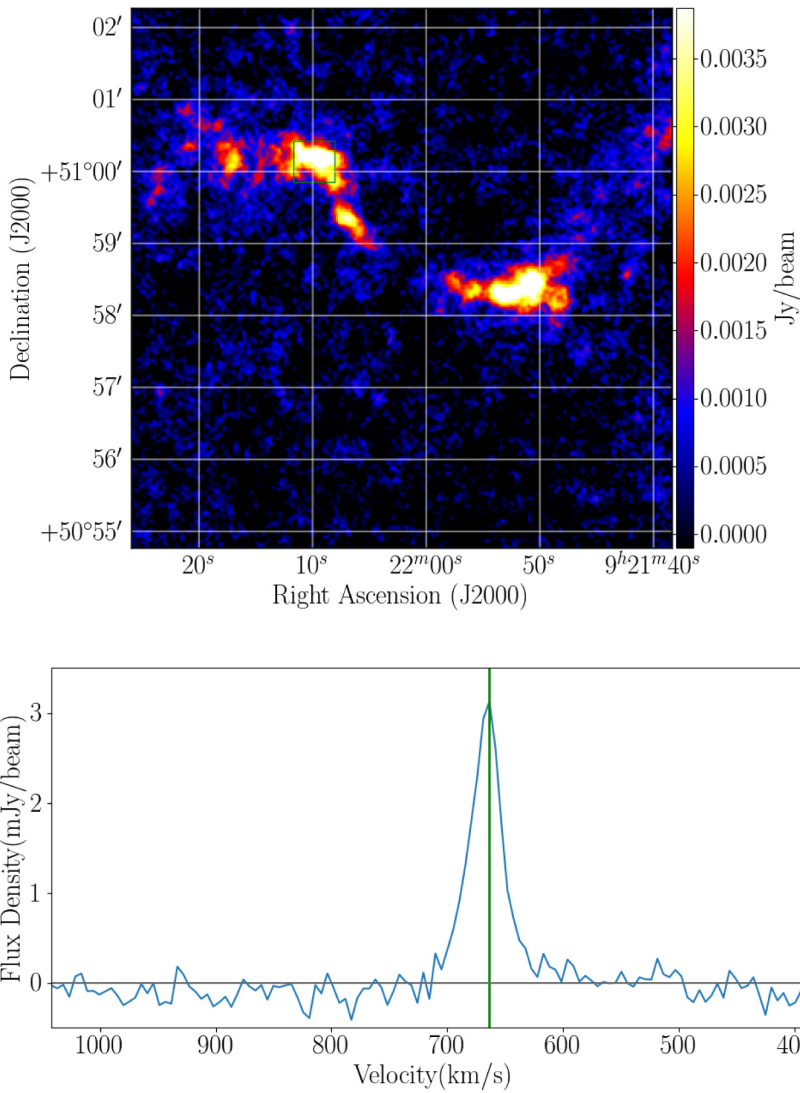
    ax1 = aplpy.FITSfigure(filename, dimensions=[0,1],
                           slices=[i,0],
                           figure=fig, subplot=[0.25,0.5,0.60,0.45])

    # degrees
    ax1.recenter(140.50924, 50.975394, width=0.125, height=0.125)
    ax1.show_colorscale(cmap=customcmap,
                       vmin=-.0001, vmax=0.00387265)

    ax1.add_grid()
    ax1.axis_labels.set_xtext('Right Ascension (J2000)')
    ax1.axis_labels.set_ytext('Declination (J2000)')
    ax1.set_tick_labels_format(xformat='hh:mm:ss', yformat='dd:mm')
    ax1.set_tick_color('white')
    ax1.set_tick_labels_style('latex')
    ax1.set_labels_latex(True)
    ax1.show_rectangles(140.54131,51.002304,0.009504,0.009504,
                       edgecolor='green', facecolor='none', lw=1)
    ax1.show_colorbar()
    ax1.colorbar.set_axis_label_text('{Jy/beam}')

    ax2 = fig.add_axes([0.2,0.1,0.74,0.3])
    ax2.plot(velocity, spectrum)
    ax2.axvline(x=velocity[i],linewidth=2, color='green')
    ax2.axhline(y=0,color='grey')
    ax2.set_xlabel('Velocity (km/s)')
```

<sup>8</sup><https://aplpy.github.io/>



**Figure 6.5.** Data cube cross section and neutral hydrogen (HI) spectrum of Galaxy NGC 2841. Data are from the NRAO Very Large Array. The box in the top panel highlights where the HI spectrum is averaged. The Python code shows the reader how to create an animation moving through the frequency or Doppler shifted velocity dimension of the data cube.

```
ax2.set_ylabel('Flux Density (mJy/beam)')
ax2.axis([max(velocity),min(velocity),-0.5,3.5])

fig.canvas.draw()
plt.savefig('pngs/'+str(i).zfill(3)+' .png', pad_inches=0)
```

## Exercises

1. One may not only just animate a data cube through frequency space. Perform the same exercise but move through the Right Ascension or Declination axis of the data cube to see the dynamical structure of the Galaxy.

## References

- [1] Astropy Collaboration; Robitaille T P *et al* 2013 Astropy: A community Python package for astronomy *Astron. Astrophys.* **558** A33
- [2] Bolton S J *et al* 2017 Jupiter's interior and deep atmosphere: The initial pole-to-pole passes with the Juno spacecraft *Science* **356** 821–5
- [3] Braun R, Gull S F and Perley R A 1987 Physical process which shapes Cassiopeia A *Nature* **327** 395–8
- [4] Connerney J E P *et al* 2017 Jupiter's magnetosphere and aurorae observed by the Juno spacecraft during its first polar orbits *Science* **356** 826–32
- [5] Finkbeiner D P 2003 A full-sky H $\alpha$  template for microwave foreground prediction *ApJ Suppl.* **146** 407–15
- [6] Robitaille T *et al* 2013 *Astron. Astrophys.* **558** A33
- [7] Hanisch R J, Farris A, Greisen E W, Pence W D, Schlesinger B M, Teuben P J, Thompson R W and Warnock A III 2001 Definition of the flexible image transport system (FITS) *Astron. Astrophys.* **376** 359–80
- [8] Haslam C G T, Salter C J, Stoffel H and Wilson W E 1982 A 408 MHz all-sky continuum survey. II—The atlas of contour maps *A&A Suppl.* **47** 1
- [9] Mellinger A 2009 A Color All-Sky Panorama Image of the Milky Way *PASP* **121** 1180
- [10] Robitaille T and Bressert E 2012 APLpy: astronomical plotting library in Python. *Astrophysics Source Code Library*.
- [11] Walter F *et al* 2008 THINGS: the H I nearby galaxy survey *Astron. J.* **136** 2563–647

# Appendix A

## *Mathematica* shortcuts and help

The listing below gives important keyboard and command shortcuts that will be useful in using *Mathematica*.

---

%	Previous result
Shift-Enter	Execute input cell
ctrl `	Toggle cell group
->	Rule transform
/.	ReplaceAll Function
?name	Information on a variable or Function
??name*	Extended information and wild card search
;	Suppress output to screen
=	Wolfram Alpha input
alt .	Abort execution
ClearAll["Global*"]	Clear the Notebook session and start fresh
CTRL-L	Copy input cell
(*code comment*)	Inline code comments

---

Science and Computing with Raspberry Pi

Brian R Kent

---

## Appendix B

### Important Python modules and resources

The listing below gives important resources in learning Python with a Raspberry Pi.

---

Starting Python	<a href="http://www.pythonforbeginners.com/">http://www.pythonforbeginners.com/</a>
Dive Into Python 3	<a href="http://www.diveintopython3.net/">http://www.diveintopython3.net/</a>
AstroML	<a href="http://www.astroml.org/">http://www.astroml.org/</a>
PyCon	<a href="https://us.pycon.org">https://us.pycon.org</a>
Jupyter Notebooks	<a href="http://jupyter.org/">http://jupyter.org/</a>
Google Python	<a href="https://developers.google.com/edu/python/">https://developers.google.com/edu/python/</a>
Scipy Lectures	<a href="https://www.scipy-lectures.org/">https://www.scipy-lectures.org/</a>
Learn Python	<a href="https://www.learnpython.org/">https://www.learnpython.org/</a>
Complex Networks	<a href="http://networkx.github.io/">http://networkx.github.io/</a>
APLpy	<a href="https://aplpy.github.io/">https://aplpy.github.io/</a>

---