An Introduction to Astrophysics with Python

Stars and planets

Online at: https://doi.org/10.1088/2514-3433/ade5f6

AAS Editor in Chief

Ethan Vishniac, Johns Hopkins University, Maryland, USA

About the program:

AAS-IOP Astronomy ebooks is the official book program of the American Astronomical Society (AAS) and aims to share in depth the most fascinating areas of astronomy, astrophysics, solar physics and planetary science. The program includes publications in the following topics:

Books in the program range in level from short introductory texts on fast-moving areas, graduate and upper-level undergraduate textbooks, research monographs, and practical handbooks.

For a complete list of published and forthcoming titles, please visit iopscience.org/books/aas.

About the American Astronomical Society

The American Astronomical Society (aas.org), established 1899, is the major organization of professional astronomers in North America. The membership (~7,000) also includes physicists, mathematicians, geologists, engineers, and others whose research interests lie within the broad spectrum of subjects now comprising the contemporary astronomical sciences. The mission of the Society is to enhance and share humanity's scientific understanding of the universe.

Editorial Advisory Board

An Introduction to Astrophysics with Python

Stars and planets

James Aguirre
Department of Physics and Astronomy, University of Pennsylvania, Philadelphia, PA, USA

IOP Publishing, Bristol, UK

ISBN 978-0-7503-5467-7 (ebook)
ISBN 978-0-7503-5465-3 (print)
ISBN 978-0-7503-5468-4 (myPrint)
ISBN 978-0-7503-5466-0 (mobi)

DOI 10.1088/2514-3433/ade5f6

Version: 20251201

AAS–IOP Astronomy
ISSN 2514-3433 (online)
ISSN 2515-141X (print)

British Library Cataloguing-in-Publication Data: A catalogue record for this book is available from the British Library.

Published by IOP Publishing, wholly owned by The Institute of Physics, London

IOP Publishing, No.2 The Distillery, Glassfields, Avon Street, Bristol, BS2 0GR, UK

US Office: IOP Publishing, Inc., 190 North Independence Mall West, Suite 601, Philadelphia, PA 19106, USA

To my students, who pushed me to make this better

To my family, for their patience and support

And in memory of Ken, who inspired me

Contents

Acknowledgments

I would like to thank Masao Sako and Robyn Sanderson for their comments and help with manuscript, and Saul Kohn for being an amazing TA the first time we tried this.

Introduction

What This Book Is and Isn't

This book is intended to provide a modern introduction to some key ideas in astrophysics with an emphasis on the underlying *physics* in the astrophysics. However, it is, by design, different from most introductory texts on astrophysics. It is *not* a comprehensive introduction to the vast panoply of modern astrophysics, or an extensive reference text, for which there are many good options available. Rather, this book is intended to foreground the key concepts and ideas while only relaying as much factual information as is absolutely essential, and to move the reader as rapidly as possible to applying the ideas to solving reasonably sophisticated problems to better get a sense of modern practice in the physical sciences. As the title indicates, these problems focus on numerical calculation and `Python` as a programming language, due to `Python`'s current ubiquitous use in astrophysics.

The book reflects my own experience in teaching a course using this material for many years at the University of Pennsylvania ("Penn"). The course started out as a traditional lecture and has evolved into its present form with interactive in-class exercises and numerical work over the past decade. The course at Penn is offered as a second-year course to physical science and engineering majors who have had introductory calculus and mechanics but who have generally not had more advanced physics courses, including electrodynamics, thermodynamics and statistical mechanics, and quantum mechanics. Since astrophysics requires some concepts from all of those fields, these need to be (gently) introduced along with the astrophysical setting as well as the numerical aspects.

My goal in writing this is to address the needs of both the student and the instructor for a course like this. Specifically, I have attempted to provide clear and concise explanations to convey the information to the reader and written exercises (which are an essential part of the text!) that will hopefully provoke a useful interaction between the student and instructor, or between the reader and text. In particular, I want to show more how modern practice uses numerical calculations to address questions that are beyond what can be addressed analytically, and to emphasize that most modern physical science involves building sophisticated mathematical *models* of physical phenomena that attempt to explain both how a given process works and to provide quantitative *predictions* of observable phenomena that can be compared to experimental or observational measurements. This model building is generally complicated enough that it is not possible to write down solutions for the mathematical equations describing it in any simple form, and hence we need to use a numerical approach. Indeed, in our first major example—the relatively simple problem of two bodies interacting only via gravity problem—famously does not have an analytic closed-form solution for the position of the bodies as a function of time, and thus requires a numerical method to solve the implicit equations for this position. In addition, in order to understand how a complicated physical model actually works, we generally need to be able to vary the

input parameters and evaluate the model's behavior over a wide range of values, which necessitates a lot of computation, which we wish to push off on the computer.

In some ways, a book structured like this necessarily values method and process over factual knowledge. Having pared the background information down to the bare minimum does lead to the risk that the reader's takeaway will be that this is all there is to know. I have tried to counteract this as I explain below. There is also the fact that, to be tractable, all the models presented here are in a very real sense "wrong", and can depart in important ways from actual facts and observations. I have tried to emphasize the importance of gaining some physical intuition and expectation by simplification, while leaving open the construction of more complex (and hopefully accurate) models as something the reader will want to explore further.

Outline of the Book

The astrophysical content of the book focuses on stars and planets. For the physics content, I have restricted myself to a level accessible to students in their second year in a US college or university, and this results in what may seem at first glance a rather bland set of topics: non-relativistic Newtonian mechanics and gravity and a simplified non-relativistic approach to quantum mechanics to describe the interaction of light and matter, combined with a bit of statistical physics. Nevertheless, students should be impressed that even with these modest materials, we can still describe a tremendous variety of phenomena of interest, including the motions of a wide range of astrophysical objects, the atmospheres and interiors of planets and stars, and the main methods for detecting and characterizing extrasolar planets. The remaining physics to fully appreciate the rest of modern astrophysics is, of course, both special and general relativity, with all of its application to black holes and cosmology and a host of "exotic" phenomena. Presenting that material is beyond the scope of what can be done here and merits its own book.

The material is presented in three major blocks. The first section is designed as an introduction to `Python` in the way we will use it here: as a method to do numerical calculations, solve differential equations, and make graphs to interpret results.[1] I focus primarily on an important problem—both historically and practically—and indeed the problem that fundamentally put the physics in astrophysics: the motion of two bodies interacting only under their mutual gravitation. The astrophysical applications of this "Kepler" or "two-body" problem are widespread, from calculating the positions of planets in the solar system, to determining the masses of stars and planets, to finding and characterizing exoplanet systems. While the two-body problem represents an idealization (the gravitational interactions of any system generally involve more than two bodies), it is so useful, and the intuition we will build so powerful that, combined with what we will learn about numerically solving differential equations, it merits taking up the first third of the text. Beginning with Newton's Laws allows the reader to start from a place where they are comfortable as

[1] I introduce only as much computer science and "programming" as is necessary to get the job done.

far as the physics content, and I use this to introduce a number of important concepts in `Python`, including how to do numerical calculations, use units, define functions, and produce plots. We then become more sophisticated in developing the idea of solving differential equations (a key skill in any physical science) and develop the methods for solving two-body differential equations numerically. This allows us to study basically every part of this interaction: the position and velocity versus time, the energy and angular momentum, and dependence of all these things on the initial conditions. There are two capstone exercises associated with this section: a fully worked-out numerical calculation of orbits under various initial conditions and a qualitative exploration of how these rules govern orbital dynamics of a satellite.

The middle section changes focus to properties of light and its interaction with matter, as despite exciting advances in gravitational wave and neutrino detection, light remains the primary means by which we obtain information from the cosmos, and its interaction with matter the primary way we have of inferring the properties of the objects that emitted it or interacted with it on its journey to us. This section is considerably lighter on numerical calculations, but develops a wide range of physical concepts that are likely to be new to students. The capstone exercises here use the Saha equation to understand the ionization behavior of hydrogen in a stellar atmosphere and velocity measurement from Doppler-shifted lines in stellar atmospheres to detect exoplanets.

The third part brings together the ideas in the first two parts and is the most rich in applications. While continuing to introduce new material, almost every chapter also attempts to synthesize multiple lines of reasoning to build models for the surface temperatures of planets and the behavior of the atmospheres and interiors of stars and planets. Our introduction to the stellar main sequence is a brief one, but the capstone exercises expand the key ideas by constructing a model for absorption lines in stellar atmospheres and how the line strength varies along the main sequence, as well as indicating how to construct a model of the stellar interior, and exploring its implications with a model of the Sun.

At the end of each major part, I include a review of the major concepts and takeaways. I also provide a small survey of additional topics to emphasize that the material presented is only a small sample of what is known, and indicate some additional directions the students could take things. In particular, there is a great deal more to say about gravitational dynamics, radiative transfer, and post-main-sequence evolution of stars, to name just a few topics.

The parts of the book build on each other sequentially and are intended to be read in order. I have tried to explicitly indicate where connections between ideas and equations occur as much as possible. In working through the exercises and problems, the reader should definitely refer back to previous material!

This Book and Active Learning

The advent of the Internet sparked a revolution in the ability of anyone to find and use information. This information is widely and freely available at the touch of a keyboard, and in great detail, to anyone who cares to look. This situation has

become even more dynamic, as at the time of writing, the use and capabilities of generative AI are exploding, and it is not at all clear what the implications will be for teaching and learning. So what is the reason for reading this book, or taking a university course, if the information is already all out there?

In part, I would answer that knowledge is not something that exists externally to human minds and can simply be stored in an algorithm or on a disk. Scientific learning must be practiced, wrestled with and internalized to be truly learned. Once learned, it can be synthesized and put to all sorts of new uses and extended in ways that will continue to surprise. Given the rapid development of generative AI, almost anything said here is likely to be out-of-date almost immediately. However, we have to believe that training actual human minds to understand the universe is still important, no matter how successfully the machines are able to do it.

With the goal of continuing to educate human minds, we note that a considerable body of education research has now shown that the traditional lecture method of presentation is terribly inefficient at actually producing reliable recall of material, much less real understanding of it. Even the venerable problem set, a staple of physics classes for generations, doesn't always produce these results. This was brought home to me by the (very bright!) student who commented at the end of the semester: "The one issue with this course was that I and others could complete the homework without having any idea what was actually going on." Clearly, this was not the desired outcome!

To address this, I began adopting an approach called Structured, Active, In-class Learning (SAIL).[2] I have used this "SAIL" approach with variations on the material in this book four times over the last decade. The idea was that each class session would be organized around an extended exercise, here included as the final section of each chapter with each chapter corresponding roughly to one class period. I have deliberately chosen to present only as much material as I have successfully presented in a one-semester course.[3] As noted above, this is justified by the need to introduce both computational and physical concepts in addition to astrophysics. The idea is that the material in the chapter is read and some reading questions completed prior to the class meeting. The instructor can then review important points or misconceptions that arise from the reading questions. The bulk of the class time is spent on the exercises, which the students work through in small groups (I have found that groups of 3–4 students are optimal) and which are overseen by the instructor and teaching assistants. This discussion—which cannot be reproduced in a book—is often the most productive part of the learning experience. If you are reading this book on your own, you will certainly learn something from the text itself, but most of the learning will take place in working through the exercises.

In writing the exercises, I strove to structure them in a way suitable to the classroom environment. First, some background is required in order to understand what questions about the Universe might be interesting. Second, this background

[2] This method goes by different names at different institutions, and emphasizes different approaches in the classroom, but all share the common feature of interactive rather than passive learning.
[3] At Penn, the semester lasts 14 weeks with approximately 28 class meetings.

attempts to lead naturally into a specific question one might ask. Generally, such a question, while interesting, is designed to be sufficiently difficult that students will not immediately know how to answer it. This leads to the third step, a bit of brainstorming and an outline of how to tackle the problem. It is important to do this step and not just immediately move into a prescribed set up of steps for solving the problem, lest the students not understand the underlying motivation and see the problem-solving process purely as a recipe. The lion's share of the time will, of course, be spent on actually solving the problem once the line of attack is understood, but then, once the problem has been mostly solved, it is important to take stock and assess what has been learned. In some cases, this will be specific skills, but in general we want to see what appreciation for the original question has been gained by working through the problem. This is the point at which one should ask "What did you learn? What was surprising? What's the big takeaway?"

This process is intended to teach "practice appropriate to the field"; that is, not just explain the results, but explain the way practicing astrophysics ask questions and think about and solve problems. This includes technical aspects like parts of physics, and also techniques like numerical problem solving. Another is engaging in activities that are like what professionals in the field do: sophisticated, multi-part problem solving; discussion of results and methods with peers; and the interpretation of results. It also addresses the fundamental problem in teaching physics that the student's quote notes: the lack of connection between theoretical constructs and mathematics on the one hand and the physical world (the real objects and phenomena) they are supposed to describe on the other. My hope is that this book will help you develop the ability to understand the way in which such mathematical explanations actually do explain the features of the observed universe.

The Role of the Computer

An important element of my approach in this book is to integrate the computer into the material and demonstrate how to think about scientific problems numerically, in addition to the traditional analytical approach, and to make numerical calculations an essential component. This attempts to address what I perceive to be a continuing gap between the actual practice of physics and the present education of our physics majors. The problems in this book use both real and synthetic data, and quantitative interpretation of its meaning. The numerical topics addressed included numerical differentiation and integration, root finding, visualizing and plotting data, and computing simple solutions to differential equations. The approach taken here is to start with simple programming features and slowly build up complexity by examples. `Python` is a fairly user-friendly language with a wide base of support online.

In spite of the centrality of the code, it is important for the reader to keep in mind that the computer is intended as an *aid* in solving problems; if you are finding it to be a hindrance, it is good to step back and think about how one would go about solving the problem if a computer were not involved.

Other Features of the Book

There are two other features of this book that merit a brief comment: the choice of how to present mathematical derivations and the use of figures. In my experience, the value of presenting derivations, either on a chalkboard or as long sections of text to be read, is decidedly mixed. On the one hand, it is highly desirable that students learn that the "boxed equations" are not handed down from on high, but come about via a process of physical reasoning, mathematical manipulation, and a clear view of an appropriate physical interpretation of the mathematical symbols. In this view, the presentation of a derivation highlights both the method by which the equations are arrived at and the numerous assumptions and approximations on which every equation relies. Moreover, correctly deriving new equations from known ones is a skill that should absolutely be developed in a student of the physical sciences, and helping students develop this skill is absolutely of pedagogical value. On the other hand, the amount of detail required to make the derivations actually clear without requiring inordinate work on the part of the reader requires a considerable amount of space and can dissipate the logical flow of a presentation, causing one to lose the forest for the trees. It is also a skill best learned through practice rather than example. Here, I have tried to have it both ways: I generally include the derivations of important equations to show the method and the reasoning where I think it is helpful to the discussion and a student's understanding but have foregone them when the derivation leads too far afield, requires concepts that are unlikely to be known to the reader, or are not going to be used again in this book, or in cases where the derivation is sufficiently lengthy that it would detract from the overall presentation. (In one particular case (blackbody radiation) I have relegated a more extensive derivation to Appendix A.) This has meant that some equations, like the Saha equation, are presented essentially by fiat, which is undesirable. Hopefully in cases where the derivation of equations is unsatisfactory to the reader or an instructor, the relevant details can easily be found and filled in from other sources.

An important final feature of this book is the figures. I have striven to use real data or calculations in producing the figures, rather than relying on schematics or cartoons. As one of the skills a reader should learn from this book is the production of high-quality, clear figures, many of the exercises involve producing figures which are highly beneficial to a full understanding of the material. All figures in the text (which are not astronomical images in the public domain) are ones which are possible to generate using the knowledge of astrophysics and `Python` contained in the book. Indeed, the `Python` code necessary is publicly available.

Author biography

James Aguirre

James Aguirre has been a professor at the University of Pennsylvania since 2008. He received undergraduate degrees from Georgia Tech in Physics and Applied Mathemathics, and received his PhD in Physics from the University of Chicago. He was a postdoc and an National Radio Astronomy Observatory (NRAO) Jansky Fellow at the University of Colorado, Boulder. He is broadly interested in the formation and evolution of galaxies and the interplay between galaxies and the large-scale structure of the universe. His research focuses on the use of radio and far-infrared wavelength telescopes to study galaxies and he has worked on a variety of experiments, including several using high-altitude balloons to carry telescopes above the Earth's atmosphere.

This is his first book.

Part I

Newton's Gravity and Orbits

AAS | IOP Astronomy

Chapter 1

A Review of Newton's Laws and an Introduction to Python

This chapter reviews Newton's three laws of motion, and his law of gravity, which form the foundation for studying the motions of planets and orbiting bodies. We also introduce the `Python` computing language and `Jupyter` notebooks and show how they can be used to make numerical calculations and keep track of units in physics problems using `astropy` units.

1.1 Overview

To begin our study of astrophysics, we'll begin with a little refresher on Newton's Laws (particularly the Second Law), and the SI (a.k.a metric) units associated with quantities like force and momentum. We'll introduce `Python` as a way to do calculations, make simple plots and keep track of units. Then we'll talk about how to think about mathematical functions as both operations a computer can do, and as arrays of numbers the computer can store, and in general how to make calculations less tedious than pencil-and-paper and calculator. In subsequent chapters, the material is presented first and then there is a follow-on exercise; this chapter is different in that the exercise is integrated with the reading.

1.2 Scientific Background

We'll start by recalling that Newton gave three laws of motion, roughly paraphrased as

1. An object in uniform straight-line motion or at rest stays in that state of motion unless an external force is applied.

doi:10.1088/2514-3433/ade5f6ch1

2. A force applied to a single particle of mass m causes an acceleration according to

$$\mathbf{F} = m\mathbf{a} = m\frac{d^2\mathbf{x}}{dt^2} = m\frac{d\mathbf{v}}{dt} = \frac{d(m\mathbf{v})}{dt} = \frac{d\mathbf{p}}{dt} \tag{1.1}$$

   where, in the modern notation, the force $\mathbf{F}$, acceleration $\mathbf{a}$, velocity $\mathbf{v}$, momentum $\mathbf{p}$, and position of the particle $\mathbf{x}$ are three-dimensional vectors that are functions of time t, and the Second Law is equivalent to a differential equation for the position as a function of time of the particle $\mathbf{x}(t)$.
3. If two bodies exert forces on each other, the force exerted by the first body on the second is equal and opposite to the force exerted by the second body on the first.

Newton also gave a specific law for the gravitational force between two objects, which will turn out to be paramount in astrophysics: gravity is only one of two long-range forces, and often a dominant force in astrophysical problems. The form of Newton's Law of Gravitation for the force exerted by particle 1 of mass m_1 on particle 2 of mass m_2, with the particles located at positions $\mathbf{r}_1$ and $\mathbf{r}_2$, is

$$\mathbf{F}_{21} = G\frac{m_1 m_2}{r^2}\hat{\mathbf{r}} \tag{1.2}$$

where $\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2$ and $\mathbf{F}_{12} = -\mathbf{F}_{21}$.

For the rest of this chapter, we'll ignore the vector nature of Newton's laws (though we'll come back to it) and just think about how to do calculations with physical quantities at all using the computer.

1.3 Using `Python` as a Calculator

We'll start by using `Python` as a glorified scientific calculator.[1] We'll use a nice web-based interface to `Python` called `jupyter notebook`. Start this up by clicking on the icon in the start menu or launcher or by typing `jupyter notebook` at the command line.

You'll need to make sure you are saving your notebook someplace sensible, so when the web browser first opens up, navigate to a folder where you'd like to save the results (or create a new one) and then select New → `Python` 3. Then rename the notebook from "Untitled" to something you'll remember (like "IntroductionToPython"). Note that the notebook is saved with a `.ipynb` extension. Verify that you can find where on your computer this was saved.

There is immediately a place to type, called a "cell" and we could start entering code straightaway. But let's go to the menu and select Cell → Cell Type → Markdown, which now makes a cell where we can type ordinary text (or math, if you happen to know LaTeX). You should use these markdown cells to add

[1] It can also be a graphing calculator, which we'll come to shortly.

annotations to your notebook, and in particular to answer the questions enumerated below. In the first cell, enter the names of the members of your group, and then type Shift-Enter (together).

In the next cell, let's now do some calculations: type

```
3+3
```

and then run the cell by using the right arrow or typing Shift-Enter. (Note that typing Enter just moves to a new line.) Does the result look reasonable? How is this different from

```
3+3.
```

(Note the extra dot. Its meaning is subtle; can you work it out?) What about

```
3*3
```

and

```
3/3
```

and

```
4/3
```

Great, so `Python` knows about arithmetic.[2] It also has some rules for real (really, rational) numbers and integers.

Another useful thing is to be able to use scientific notation. How do we enter a number like 3×10^{12}? There are actually several ways to do this. We can type

```
3*10**12
```

Notice how we're expressing exponentiation. We could also type

```
3e12
```

Notice that this has the same meaning. Also, notice that "e" here is *not* the Euler constant e, but a stand-in for "times ten to the power following the e". This can be kind of confusing at first, but the compact notation is useful.

So, what about trig functions?

Question 1. What happens when you enter `sin(1.)`? Look carefully at the output and try to explain what `Python` is telling you. Explain this is in a Markdown cell.

[2] Hopefully this makes clear what symbols `Python` is using for addition, multiplication and division. What about subtraction?

Well, that's really going to be a problem for a(n) (astro)physics course! But we have learned something important: Python tries to tell us what's wrong, and in general you should start at the last line of gobbledy-gook to start decoding the problem.

Python by itself has very limited capability for doing scientific computing, but fortunately, many people have written additional software, called "modules"[3] for it. For now, we will issue the cryptic command

```
import numpy as np
```

and now we can do

```
np.sin(1.)
```

Basically, this tells Python that it should look for the sin function in the numerical Python (numpy) module, which we're going to call np for short.

Notice that we can also make assignments of the output to variables to save for later, so that

```
a = np.sin(np.radians(45))
```

As usual, the argument of sin needs to be in radians, but numpy provides a convenient method for doing the conversion. Also notice that "=" doesn't mean a logical test for equality, but rather "assign the value on the right hand side the name on the left."[4] Notice that running this cell doesn't seem to do anything, but if I now say

```
print(a)
```

something interesting appears.

Question 2. What did you expect for the answer here? Notice that numpy is directly evaluating the numerical value, and not expressing it as you might, with square roots. Can you figure out how to get numpy to evaluate what you know to be the right answer from trig? A helpful thing to know here is that typing np. and the "Tab" key will show possible completions for numpy functions.

[3] Don't worry about what a module actually "is" just yet. For now, think of it as some additional functionality we're going to add to bare Python, and that we need to request specifically.

[4] This use of the "=" to mean assignment rather than mathematical equality (which for example, is the same regardless of which side of the = things appear on), is quite old in computer science, and isn't changing now. It may take some getting used to, or you might not find it confusing at all.

1.4 Arrays and (Avoiding) Loops

You may have learned in another course on programming about the all-important for loop. And you may be tempted to use for loops to repeat calculations for many different values, but in Python we can mostly use *arrays* (lists of numbers stored in a single variable) instead: most common functions are defined such that if you feed an array to the function instead of a single value, it will *automatically* loop through the values in the array. It is very rare that you actually have to write a for loop in Python!

For example, I want to calculate values of the cos function for a bunch of different input arguments. Since doing many tedious calculations is what we have computers for, let's consider what's necessary. First, I need an independent variable x that runs over some range (say one period), and a variable to hold $\cos(x)$ for every value of x. These two variables are examples of numpy arrays, a data type defined in the numpy module. We can create them as (type along here):

```
x = np.linspace(0,2*np.pi,num=100)
y = np.cos(x)
```

The function np.linspace creates an array of 100 evenly spaced values of x between 0 and 2π, and y (created by calling the cosine function with x) now holds the values of the cosine for each value of x. The cosine function is an example of a function that will accept either a variable holding a single number or an array, and return something with the same type as its argument. If we did it right, y should now also be an array with 100 elements, in the same order as x. You can verify this, since numpy arrays have a method to tell you their "shape", in this case, how many elements:

```
print(y.shape)
```

You can ask for all the values of y by typing

```
print(y)
```

or you can ask for any specific one by saying

```
print(y[15])
```

which will give the 16th in the list (we number from 0 in Python). The number 15 in the example is called an *index*, an integer that specifies the position of a value in the array. Every value in the array has an index. We can also get a "slice" of the array, which is some subset of it defined by a range or list of indices. The : operator is shorthand for specifying a range; this range includes every value between the first one (inclusive) and the last one (**exclusive**). What does, for example,

```
print(y[15:18])
```

produce?
Now try

```
print(y[15:18:2])
```

and

```
print(y[18:15:-1])
```

and see what you get!

Often we want to find a specific number in a list of numbers; we can use various functions to figure out the index of a specific number.

Question 3. Suppose we want to find the index of the array element that is the largest. We can use `np.argmax` to do this. At what indices does our `y` = cos `x` array have a maximum?

It would be much more interesting to plot this than to look at the list of numbers, and here we need another module:

```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.xlabel('x [dimensionless]')
plt.ylabel('y [dimensionless]')
plt.show()
```

The basic logic of plotting is to create the plot, add annotations, additional curves, and then at the end show it with `plt.show()`. The above should let you see a plot of the cosine.

Question 4. Now calculate $\sin(x)$ for the x values you have defined. Add it to the plot of the cosine (just add another `plt.plot` line in the same cell). Also calculate $\sin(x)^2 + \cos(x)^2 - 1$ and plot it on the same plot. (If you already know something about `Python` or other coding languages, you might be tempted to do something here involving a `for` loop or similar. Resist the urge! You can write the above expression very much like you would in a math course, and it will be just fine. Remember, many functions are defined for arrays as well as individual numbers!) You can raise numbers (or each element of a list of numbers) to a power in `numpy` using either the `np.power` function (defined for any power including non-integers) or the `**` notation from earlier (defined only for integer powers, but faster than `np.power` when you can use it). Here, use `np.power(np.cos(x),2)` to square each element of $\cos(x)$, in order to try it out. Don't forget to label your axes!

1.5 Using Python Modules to Help with Units

You've probably had it drilled into you that physical quantities always have a magnitude and a unit.[5] And you've also learned that units obey algebra as well: products and quotients of units make new units, and sums of dissimilar units are nonsense. Further, when taking a sin or exp or ln of something, the argument should be dimensionless.

The structure of Python makes it possible to glue numbers and their units together, and to do sensible things with their arithmetic combinations. A very nice implementation of this is done in the module astropy, which as the name implies, has lots of nice astronomy-related features and functions.

In a new cell, type

```
from astropy import constants as c
from astropy import units as u
```

Now we have units available to us from the u module, and physical constants (including useful astronomical ones) from c. So now

```
x = 1.*u.meter
```

defines *x* as a "Quantity object" equal to 1 meter (value and unit!). An *object* is another computer science concept. In brief, objects have both data and some defined operations you can do on the data (its *methods*.) The data associated with objects are referred to as its *attributes*, and are not limited to the value of the variable. For example, the shape of a numpy array is one of its attributes, which you accessed earlier. Nearly everything in Python is, or can be thought of as, an object, including the arrays you already defined—and the modules you loaded!

Similarly, if we define

```
t = 5.*u.second
```

notice that

```
v = x/t
```

defines a new Quantity with the right units (and we had to do nothing!). Now, you can see that this new capability is useful in a number of ways:

- You can keep track of the units in your calculations easily
- You can avoid mistakes with units
- There is even a way to do conversions between units[6]

[5] And this book will be no exception!

[6] This is one of the operations, or methods, defined for the Quantity object.

Question 5. Let's see that Point 1.5 is true. What happens if you type `x + t`? (It will look like `Python` got very angry, but remember to scroll to the last message.) Briefly explain why this makes sense.

Objects are really what `Python` is all about, and while I won't go into detail, you can start to see how they are useful by noting that the thing we called `v` isn't just a number, but it has units (another example of an attribute), and moreover, you can *do* things with it. So regarding Point 3, Quantity objects come with a useful method called `to`, which allows conversions:

```
v.to(u.kilometer/u.second)
```

Question 6. What happens when you type `x.to(u.second)`? What about `x.to(u.imperial.mile)`?Again, explain why this makes sense.

Wow, `Python` is going to save us tons of time and errors!

Now, we can also use those physical constants, all of which are Quantity objects, to do something like calculate a force:

```
m = 1.*u.kilogram
r = 1.*u.meter
F = c.G * np.power(m,2)/np.power(r,2)
```

Notice that we've indicated the function of raising to a power with a somewhat long notation.

Question 7. What is the value of `F` above in Newtons?

A subtle point is in order about the `to` method: if you type the last line at the command prompt, you will get back F converted to Newtons. However, if you ask about F again, you will find its units are still reported as kg m s^{-2}, and not converted. To get the F variable to hold the new unit, you must reassign it

```
F = F.to(u.Newton)
```

In this case, the conversion is kind of silly, because we're savvy enough to know this is actually the *definition* of the Newton, but consider

```
F = F.to(u.picoNewton)
```

Now F will appear in pN. It's important to notice that `astropy` is fairly smart about correctly combining units, but they're not always in the form you'd like. Consider

```
A = np.power(10.*u.centimeter, 2)
P = F/A
```

Now, P is actually a pressure (a force per area), but if you ask about it, you get pN cm^{-2}, and perhaps you'd prefer a pressure unit like the Pascal (Pa):

```
P.to(u.Pascal)
```

or the ever-popular pound per square inch (psi):

```
P.to(u.imperial.psi)
```

1.6 Defining Your Own Functions

Now that we've seen how to get the computer to take some of the tedium out of arithmetic and unit conversion, let's take the tedium out of computing quantities over and over, like the gravitational force. We want to think of the force as a function, where you specify the two masses and the distance between them (the *arguments* of the function), and get back the [magnitude of the] force (what the function *returns*).

`Python` lets us define functions with a notation that looks a lot like mathematical notation:

```
def grav_force(m1, m2, r):
    F = c.G * m1 * m2 / np.power(r,2)
    return F
```

Here the masses m_1, m_2 and the distance between them r are the inputs (named, naturally, `m1`, `m2`, and `r`, though you could in principle name them anything you want) and F is the value returned by the function, which is itself named (`def`ined to be) `grav_force`. G is a constant of nature, so we don't require the user to enter it, and we make use of the fact that `astropy` has already stored it for our use. If we wanted this to be a stand-alone function we would need to ensure that `astropy` and `numpy` were imported before calculating F (you can, for example, import modules inside functions). As written, our function does not require that m_1, m_2, and r to have

units, but if they do, F will have the correct dimensions. The equivalent mathematical notation would be

$$F(m_1, m_2, r) = G\frac{m_1 m_2}{r^2} \tag{1.3}$$

It's important to understand that, just like our mathematical function Equation 1.3, the `Python` function `grav_force` is an abstract object: it *could* be calculated for any values of its input, but as it stands it is pure possibility: no actual numbers have been computed. Likewise, the variable `F` that we defined inside the function `grav_force` is only defined while the function is being executed—if you try to print the value of `F` after defining and running `grav_force`, you'll get an error (unless you assigned the value returned by `grav_force` to a variable called `F`).

To actually use this function, we need to pick a set of values for which we want to evaluate it, and then also have a place to store the result

Let's try it!

Question 8. Compare the gravitational force of Jupiter (in Newtons) on you to the gravitational force of a member of your group. State explicitly your assumptions about the mass and distance of the objects. It is helpful to know there is a constant `c.M_jup` for the mass of Jupiter, and that distance to Jupiter is roughly given by `5.*u.AU`, that is, about 5 astronomical units. (How far is that? Convert it to meters!)

Full list of references

Chapter 7

Reed, B. C. 2023, Keplerian Ellipses 2nd edn (Bristol: IOP Publishing)

Chapter 21

Carroll, B. W., & Ostlie, D. A. 2017, An Introduction to Modern Astrophysics (2nd edn; Cambridge: Cambridge Univ. Press)
Demircan, O., & Kahraman, G. 1991, Ap&SS, 181, 313

Chapter 22

Husser, T. O., Wende-von Berg, S., Dreizler, S., et al. 2013, A&A, 553, A6

Chapter 23

Bahcall, J. N., Pinsonneault, M. H., & Basu, S. 2001, ApJ, 555, 990