Contents

1 Course introduction ................................. 6
   1.1 Physics studies ................................ 7
   1.2 Coding techniques .............................. 7
   1.3 Numerical methods .............................. 7

2 Course guidelines: structure and expectations .............................................. 8
   2.1 Learning outcomes ................................ 8
   2.2 Teaching and assessment ........................... 8
   2.3 PH3080/PH3082 .................................... 8
   2.4 Examination ........................................ 8
   2.5 Continuous Assessment ............................ 9
   2.6 Module attainment strategy ....................... 9
   2.7 In-class questions (ICQs) .......................... 9
   2.8 Homework tasks (HWTs) ............................ 9
   2.9 Submission penalties .............................. 10
   2.10 Collaboration and plagiarism ..................... 10
   2.11 Practice exam ..................................... 11

3 Timetable .............................................. 11

4 Mathematica introduction - PC classroom sessions - Week 1 ......................... 12
   4.1 Accessing Mathematica ............................. 12
   4.2 What does Mathematica look like? ............... 12
   4.3 Using help .......................................... 12
   4.4 Simple Calculation ................................ 13
   4.5 Text insertion ....................................... 14
   4.6 Printing .............................................. 14
   4.7 Saving ................................................. 14
   4.8 Expressions .......................................... 15
   4.9 More than a calculator .............................. 15
   4.10 Clear or Unset ...................................... 15
   4.11 Building functions ................................. 16
   4.12 Numerical value - the N function ............... 17
   4.13 Palettes and other functions ..................... 17
   4.14 Greek alphabet symbols ........................... 17
   4.15 The semicolon (;) .................................. 17
   4.16 Comments in Input cells ............................ 18
   4.17 Lists .................................................. 18
   4.18 List manipulation ................................... 19
   4.19 Plotting graphs ..................................... 20
   4.20 Rules .................................................. 22
   4.21 Manipulate and animate ............................ 23
   4.22 Scope of variables ................................. 23
   4.23 3D plotting .......................................... 24

5 Mathematica introduction - PC classroom sessions - Week 2 ......................... 26
   5.1 Differentiation ....................................... 26
   5.2 Integration ............................................ 26
   5.3 Piecewise functions ............................... 27
   5.4 Differential equation solving .................... 27
   5.5 Finding Roots ....................................... 29
5.6 Solving systems of equations

5.7 The bakery problem

5.8 Using strings

5.9 Cell formatting

5.10 Using data files

5.11 Oil pipeline cost problem

5.12 Pure functions

5.13 Programming with rules and patterns

5.14 Patterns in functions

5.15 Solving differential equations with triggering events

5.16 Solving differential equations for parametric study

5.17 Debugging methods

6 Celestial mechanics (week 3)

6.1 Programming techniques

6.2 Review of physics background

6.3 Study

6.4 Advanced problems

7 Galaxy redshift (week 4)

7.1 Programming techniques

7.2 Physics background

7.3 Study

7.4 Advanced problem

8 Double pendulum (week 5)

8.1 Programming techniques

8.2 Review of background physics

8.3 Study

8.4 Advanced problem

9 Independent learning week (week 6)

9.1 Translating programming languages

9.2 Small optional project

10 Fourier (week 7)

10.1 Review of background

10.2 Programming techniques

10.3 Study

10.4 Advanced problem

11 Quantum mechanics (week 8)

11.1 Schrödinger’s equation in 1D

11.2 Study

11.3 Advanced problem

12 Geometric optics (week 9)

12.1 Physics background

12.2 Programming techniques

12.3 Study

12.4 Advanced problem
### 13 Waves (week 10)
- 13.1 Physics background .............................................. 55
- 13.2 Programming techniques ........................................ 55
- 13.3 Study .................................................................. 56

### 14 Waves in periodic structures (week 11)
- 14.1 Physics background .............................................. 58
- 14.2 Programming techniques ........................................ 58
- 14.3 Study .................................................................. 60
- 14.4 Advanced problem ............................................... 60

### 15 Past paper (2018) .............................................. 61
1 Course introduction

What is computational physics? In short, it is the use of a computer to solve problems in physics. The general approach to this problem solving is to, first, describe the physical system mathematically. This leads to a theoretical model of the problem which can then be used to define a set of instructions for the computer (a program). In this course, we have chosen Mathematica as it provides the capabilities required, while allowing quick implementation and visualisation of solutions (see table below). This means that we can spend more time investigating physics and less on formal coding techniques.

In this module, we will study a number of physical systems where the mathematical models are given and are straightforward. Implementing these models in Mathematica allows us to analyse, synthesise and evaluate our understanding of the physics. In this way, we can get insight into the physics and debug misconceptions. Programming offers a consistent, logical framework that delivers real-time feedback on our physics comprehension.

There are other programming languages that can be used in computational physics. Many languages share similar programming structures, so a good familiarity with the underlying methods of programming in one language can often be transferred to a new language relatively easily. The learning curve for a second and subsequent languages should normally be much less than for the first. Some languages are better than others in terms of ease of setting up for different types of work, speed of processing, etc.

<table>
<thead>
<tr>
<th>Language</th>
<th>Description</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>compiled, procedural, object oriented</td>
<td>Fortran has a long heritage, can be relatively fast and efficient to run, and links easily to a wide range of numerical routines, etc. It is widely used in astrophysics in the school, and is the language used in the Computational Astrophysics and Monte Carlo Radiation Transfer Techniques modules.</td>
</tr>
<tr>
<td>Python</td>
<td>interpreted, functional, object oriented</td>
<td>Python is a newer language that is gaining in popularity. It is taught in the second year in the School of Mathematics.</td>
</tr>
<tr>
<td>C/C++</td>
<td>compiled, procedural, object oriented</td>
<td>C and its variants are widely used, particularly when dealing with instructions for microprocessors. A special version of C is used in the second year physics lab in the Arduino work.</td>
</tr>
<tr>
<td>Java</td>
<td>compiled, procedural, object oriented</td>
<td>This is currently used in first year computer science modules.</td>
</tr>
<tr>
<td>MATLAB</td>
<td>interpreted, procedural, functional</td>
<td>Matlab is primarily designed for processing large amounts of numerical data, though it can do a range of things. It is not currently taught in the School, but it is used in a number of research groups and final year projects.</td>
</tr>
<tr>
<td>LabVIEW</td>
<td>compiled, graphical, object oriented</td>
<td>This programming language is widely used to interface computers to experiments for control and data acquisition and processing. It is currently taught in four afternoons in the honours physics lab.</td>
</tr>
<tr>
<td>Mathematica</td>
<td>interpreted, procedural, functional</td>
<td>Mathematica is primarily an algebraic programming language. It includes a large library of functions that allow quick learning to the stage where it can be used to model and display results of physical systems. Mathematica is currently introduced in second year physics, and we use it in the Computational Physics module.</td>
</tr>
</tbody>
</table>
1.1 Physics studies

There are a number of studies where we show the use of Mathematica in different physical situations. Two case studies deal primarily with data treatment and visualisation (Observing the galactic Doppler shift, Fourier transform and filtering). The rest of the case studies look at optics (ray tracing and waves), classical mechanics (planet motion, chaotic pendulum motion), and quantum mechanics (solving Schrödinger’s equation).

1.2 Coding techniques

As we progress in computational physics, we will learn and consolidate a number of coding techniques. Among others, we will define functions/subroutines, use loops, manipulate arrays, create conditional statements, import, extract and plot data. These techniques are at the heart of all programming languages and what we will learn in Mathematica is transferable to other programming languages such as the ones mentioned in the table.

1.3 Numerical methods

To model physical systems we need to use numerical methods. Mathematica forms a powerful interface to functions implementing high performance numerical methods such as numerical integration, RungeKutta and finite-elements methods for solving differential equations, and Fourier transform to study periodic signals.
2 Course guidelines: structure and expectations

2.1 Learning outcomes

After successfully completing this module you will be:

- Able to use Mathematica to solve and visualise problems in physics and astronomy.
- Competent in the use of Mathematica’s Help system to allow you to do more.
- Empowered to explore further application of Mathematica and its more advanced features.  

2.2 Teaching and assessment

We will introduce the module with a lecture during orientation week. The main teaching for this module is then through mentored PC classroom sessions. The weekly structure of the course is as follows:

- In the first two weeks of the semester you will have an "Introduction to Mathematica". These sessions will give you a grounding in Mathematica’s syntax and methodology. There will be some application of Mathematica, but the main thrust of the introduction is for you to become familiar with the Mathematica environment, programming structures, and its comprehensive Help system. (2 introduction weeks)
- In weeks 3-11 (exc. 6 ILW), you will use Mathematica to explore physics and astronomy through a series of case studies, with a new topic each week. (8 case-study weeks)

Each week you will have two scheduled 2 hour PC classroom sessions, i.e. four hours of contact time. During the case study weeks, each of these session will have a specific roll:

- The first session will introduce you to the technical programming necessary for that week’s case study. This session is not compulsory. However, we will put these programming techniques into practice, and formally assess them, in the second session.
- The second session will apply the first session’s learning to a problem in physics and astronomy. To ensure that you have engaged with the first session material, the second session will start with a short in-class question (ICQ), in which you will apply your first session learning. This question has one percentage point attached. Attendance at this second session is compulsory; an Academic Alert will be issues for any unauthorised absence.

We have a similar assessment strategy in weeks 1 & 2, whereby time is set aside in the second session for an ICQ. In week one there will be no formal mark attached, this allows you to see the assessment in action without the stress of it being associated with a mark.

Assessment for this module is split 75:25 between examination and continuous assessment:

2.3 PH3080/PH3082

These notes also apply to students taking the PH3080 Computational Physics module and the PH3082 Mathematics for Chemistry / Physics module. PH3082 students finish with teaching on this course at the end of Week 5, and will complete a 1.5 hour class test in Week 8 (second session).

2.4 Examination

You will have a 3 hour examination in the examination diet. This accounts for 75% of your mark for the module. The examination will involve using Mathematica to solve a number of problems. During the examination, you will have access to the Mathematica Help system.

1 Particularly with a view to applying these to your other studies.
2.5 Continuous Assessment

Continuous assessment comes partly from the ICQs mentioned above (totalling 9%). The additional 16% arises from some of the weekly Homework Tasks (HWTs). The length of the HWTs will increase as you progress through the module. The total time allocated to HWTs is 10 hours, spread throughout the semester.

Assessment information and case study topic is summarised the following table.

<table>
<thead>
<tr>
<th>Week</th>
<th>Topic</th>
<th>ICQs</th>
<th>HWT</th>
<th>Open</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 hour introductory lecture</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Introduction to Mathematica</td>
<td>ICQ01-10 mins-0%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Introduction to Mathematica</td>
<td>ICQ02-10 mins-1%</td>
<td>HWT02-30min-0%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>3</td>
<td>Celestial mechanics</td>
<td>ICQ03-10 mins-1%</td>
<td>HWT03-30min-3%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>4</td>
<td>Galaxies</td>
<td>ICQ04-10 mins-1%</td>
<td>HWT04-60min-0%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>5</td>
<td>The double pendulum</td>
<td>ICQ05-10 mins-1%</td>
<td>HWT05-60min-4%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>6-ILW</td>
<td>Practice questions and/or advanced problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Fourier</td>
<td>ICQ07-10 mins-1%</td>
<td>HWT07-90min-0%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>8</td>
<td>Schrödinger’s equation</td>
<td>ICQ08-10 mins-1%</td>
<td>HWT08-90min-4%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>9</td>
<td>Ray optics</td>
<td>ICQ09-10 mins-1%</td>
<td>HWT09-120min-0%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>10</td>
<td>Waves</td>
<td>ICQ10-10 mins-1%</td>
<td>HWT10-120min-5%</td>
<td>Mon 10AM</td>
<td>Wed 6PM</td>
</tr>
<tr>
<td>11</td>
<td>Waves in periodic structures</td>
<td>ICQ11-10 mins-1%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2.6 Module attainment strategy

If you have been keeping track, you may have noticed that in total we have accounted for 41 hours of contact time, and outlined 10 hours for HWTs. For a 10 credit module, we would expect the average student to spend 100 hours working on the course, so there are 49 ‘missing’ hours. You should therefore expect to allocate time outside of that prescribed above. Learning Mathematica, and applying it to problems in physics and astronomy, is a lot like learning a new spoken language. You can be given the rudiments, but you will only become fluent when you use it habitually. Proficiency in Mathematica, and its use in solving problems, can only be achieved through practice. It is not something that you can crash learn at the last minute. Reviewing earlier learning and applying Mathematica to your other courses are two possible means of you putting in the practice that is required to become self-assured with Mathematica.

We will give you additional practice questions (PQs) and studies that you should engage with continuously and during the Independent Learning Week. We will also direct you toward other learning material. The Mathematica Help system and Wolfram’s Demonstration Projects are also great places from which to draw inspiration, and to gain insight and understanding into the use of Mathematica.

2.7 In-class questions (ICQs)

Each week, at the start of your second weekly session, you will have a question based on material covered in the first weekly session. The ICQs can only be accessed on the Physics and Astronomy PC classroom computers. You have ten minutes to answer the question, with the room being under examination conditions. As such, you are asked to enter the room quietly as some students may have started working on the question. The question opens on the hour, and closes at twenty past the hour without exception. Marks and feedback are released on the Friday evening.

2.8 Homework tasks (HWTs)

You should develop your answers in Mathematica, and once ready, they should be submit by copying and pasting specifically the necessary code from your notebook into the appropriate Moodle field. You must not submit any Mathematica output.
We expect you to use a university computer to undertake Homework. If you are using your own computer to work on any Homework then you must ensure that you have a stable environment and that your computer will not cause you any problems during the Homework time allocation. You can check your computer’s suitability to the task using the notebook available at:

https://www.st-andrews.ac.uk/physics/ph3080/CheckMathematica.nb

You are not to undertake HWTs during your scheduled PC classroom slots.

You are expressly forbidden to collaborate while undertaking the HWT. These must be done individually and independently. Any material found to be copied or adapted from others will be heavily penalised. You should refrain from discussing the Homework task until 48 hours after the submission deadline.

All HWTs with course credit attached will be marked according to the marking scheme and the above penalties. Marks are provisional and subject to moderation once all coursework has been assessed. This will allow us to account for differences in the randomly-allocated questions. All other HWTs will be used for formative assessment. An Academic Alert will be issued for any missed HWTs without prior permission.

Feedback and marks, where appropriate, will be made available within two weeks from the submission deadline.

2.9 Submission penalties

Part of the assessment of our module is in the form of weekly timed Homework task. The following penalties are in place:

**Over-time penalty:** Submissions are subject to over-time penalties in a scheme that parallels the University’s “Penalties for Work of Incorrect Length” for written coursework assignments. We use an equivalent of Scheme C, whereby the penalty is 5% of the maximum available mark for work that is over-time to any extent, then a further 5% of the maximum available mark per additional 5% over. The following table illustrates this for an illustrative 60 minute Homework. The penalty will never exceed the mark awarded for the work, i.e. no negative marks.

<table>
<thead>
<tr>
<th>Minutes over-time (t)</th>
<th>Penalty (%)</th>
<th>18 &lt; t ≤ 21</th>
<th>39 &lt; t ≤ 42</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 &lt; t ≤ 3</td>
<td>5</td>
<td>35</td>
<td>70</td>
</tr>
<tr>
<td>3 &lt; t ≤ 6</td>
<td>10</td>
<td>40</td>
<td>75</td>
</tr>
<tr>
<td>6 &lt; t ≤ 9</td>
<td>15</td>
<td>45</td>
<td>80</td>
</tr>
<tr>
<td>9 &lt; t ≤ 12</td>
<td>20</td>
<td>50</td>
<td>85</td>
</tr>
<tr>
<td>12 &lt; t ≤ 15</td>
<td>25</td>
<td>55</td>
<td>90</td>
</tr>
<tr>
<td>15 &lt; t ≤ 18</td>
<td>30</td>
<td>60</td>
<td>95</td>
</tr>
</tbody>
</table>

In the example shown, Moodle will automatically submit your work at 60 minutes late, i.e. 120 in total. The Moodle clock shows a countdown to the total available time to zero-mark submission. To avoid an overtime penalty, you should be aiming to submit before the timer reaches half of its original value. Note that after double the allocated time the penalty will have accumulated to the point where the Homework value has reduced to zero. At this point the Homework is automatically closed, submitted and zero marks will be awarded.

**Late submission penalty:** Each Homework opens on Monday morning and has a 6pm on Wednesday submission deadline. You should ensure that you submit before the deadline as there is a late submission penalty (University Variant C of the “Penalties for Late Work submission” with a cut-off after 48 hours).

**Overall penalty:** The late submission and overtime penalties are added together if your submission is late and overtime.

2.10 Collaboration and plagiarism

Working and learning together is permitted, and indeed encouraged. During the PC-classroom session you will find that if you discuss ideas with your peers then both you and they benefit. There are also
demonstrators available, and they should be consulted as required. If you cannot solve a problem by yourself in five minutes, then you should be seeking assistance. The express exception to collaborative working is when you are working on assessed activities, i.e. ICQs or Homework tasks, and of course the final examination. In all of these instances you are expected to work entirely independently. As your Homework does not take place in a controlled environment, you need to take greatest care to do these individually and independently. Any material found to be copied or adapted from others will be very heavily penalised.

2.11 Practice exam

Past exam paper and example solution will be available on moodle.

3 Timetable

<table>
<thead>
<tr>
<th>Time</th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:00 - 14:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15:00 - 16:00</td>
<td>Technical PCC</td>
<td>Technical PCC</td>
<td></td>
<td>Application PCC</td>
<td>Application PCC</td>
</tr>
<tr>
<td>16:00 - 17:00</td>
<td>Group A2 16:05-17:55</td>
<td>Group A4 16:05-17:55</td>
<td></td>
<td>Group B2 16:05-17:55</td>
<td>Group B4 16:05-17:55</td>
</tr>
<tr>
<td>17:00 - 18:00</td>
<td>Technical PCC</td>
<td>Technical PCC</td>
<td></td>
<td>Application PCC</td>
<td>Application PCC</td>
</tr>
<tr>
<td>18:00 - 19:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 Mathematica introduction - PC classroom sessions - Week 1

Our Industrial Advisory Committee made recommendation that our students should be familiar with a mathematical programming language; the ability to use Mathematica, or a similar high-level mathematical modelling tools, is a very valuable transferable-skill for all physicists and astronomers.

4.1 Accessing Mathematica

For our course we have standardised on Mathematica version 11.0. The University of St Andrews has a site licence for Mathematica, meaning that it can be installed on any machine owned by the university. You will find Mathematica in the university’s PC classrooms (launch it from Apps Anywhere). Useful nearby locations include the Physics and Astronomy (PandA) PC classroom, the Maths classroom across the link-bridge to Maths, and the Wedge cluster, across the link-bridge to the Biomedical Building.

The University’s site licence also allows students to install Mathematica on their own computer. You can find the “Wolfram Mathematica access request” link, in the “Student Resources” section of our “Staff and Student” web page. You should create a Wolfram account using your university email address; in that way it will associate with the University’s site licence. You can then download and activate Mathematica on your own computer. It is a large download (~ 4GB).

4.2 What does Mathematica look like?

Having launched Mathematica, and then chosen “New Document” on the “Welcome Screen”, you will see the main user interface, called the Mathematica notebook. There is another, invisible, component to Mathematica, called the kernel. The kernel is Mathematica’s computational engine. This is where all calculation are performed and values are stored.

Looking at the notebook window, you should note the 100% at the bottom-right corner. This zoom facility allows you to change the size of the notebook content. Make sure you zoom to a point where you can distinguish a full stop from a comma (.,), since, as with any programming language, Mathematica is very particular on exact syntax!

4.3 Using help

Mathematica has an excellent help facility. The most convenient way of using these is to press the F1 key on the keyboard. If the insertion point is on a blank portion of the notebook, then the “Documentation Center” will open. The documentation centre gives broad category headings, under which you will find a host of useful information. However, if you press F1 while the insertion point is adjacent to some characters in the notebook, then Mathematica will direct you to help pertinent to that expression. Notice the spelling of the word “center” in “Document Center”; American spelling is used throughout Mathematica.

My personal methodology in looking for Help in Mathematica is to try to express in mathematical terms what it is I’m trying to get Mathematica to do. I then enter a function name (or mathematical concept) that I think is similar in nature. Mathematica has excellent cross-linking between help topics, so when I press F1 and discover that the page does not do exactly what I want, I use the “See Also”, “Tutorials” or “Related Guides” sections for related information. This will generally give me a steer in the right direction. If not then I try to re-frame the initial help query and try again.
For example, if I’m told to select an element from a list, then I’d type in the word “list” and press $F_1$. This will take me to a page of search results. Notice that it does not go directly to the List function as the search query was not capitalised . . . more on capitalisation later. In the returned search, I read down and discover the third item is “List Manipulation” - sounds useful (you should be trying this by-the-way). Opening that, I discover a host of functions for working with lists. Reading through I find the Part function which seems to fit the bill.

I find that effective use of the well-indexed help system is key in getting things done in Mathematica.

One word of caution. In the university PC classrooms the key combination $F_6-F_1$ will shut down the computer. Please do not use this key combination while using any of the university computers!

4.4 Simple Calculation

When you start typing in Mathematica, it treats your typing as mathematical input by default. Open a new notebook (File Menu | New | Notebook) and type $2+3$. To make Mathematica evaluate this input you then press $\text{SHIFT}+\text{ENTER}$. This passes the input expression to the kernel, and the kernel passes back the result as an output.

Worthwhile noting:

- Mathematica added $\text{In}[1]:=$ to the input expression, and $\text{Out}[1]=\text{to the kernel return.}$
- Mathematica also displays a “Suggestions Bar” under the last output, giving possible next steps for your calculations. I’m not sure how useful this might be, it reminds me a bit of the paper-clip that Microsoft used to have in Word. You can switch this option off in the Edit | Preferences menu, though in the PC classroom this may revert to being on with your next login.
- Mathematica added cell braces on the right-hand-side of the new notebook. These cell braces are the key to formatting content in Mathematica. More on formatting in the text insertion section later.
- The kernel is shared between all open notebooks, and expressions go into what is called the “Global” context by default. It is possible to isolate notebooks from each other. Indeed, **the Mathematica help system is a sequence of editable and executable notebooks**, but they run in their own context so that their symbols and expressions don’t interfere with your work. You can edit the help content to examine is working, it will not save your changes, so feel free to tinker around with the examples.

Exercises 0

1. As you work through this material you should be trying out everything that is discussed; even when not labelled explicitly as an exercise.
2. As you encounter a new function you should read its help entry, paying particular attention to the examples. In recent years, students have found it worthwhile to compile a list of the functions they encounter, with a sentence about each. At the end of each of these introductory worksheets we will give a list of functions first met in the session.
4.5 Text insertion

As with any computer programming, it is generally easy for you to understand the notebook as you build it. However, at some later time, you, or someone else reading your notebook, may have difficulty following what is going on. It is useful then to provide titles and comments throughout your notebook; at least a title for the notebook, and a text cell with your name is essential.

In your new notebook move the mouse to just above the 2+3 input cell until the, usually I shaped, insertion cursor changes to a horizontal I. Click to place the insertion point. We want to add a Title cell, so from the Format Menu, choose Style, then Title. Now type “My first notebook”. Immediately below this insert a new cell with style “Text” and type your name and the date.

When styling your cells, you should try to set the style prior to creating the cell, as we did above. Alternative, if you want to change the style of a cell then you should make sure that you select the cell bracket (on the right) prior to changing style. It is possible to change the style of just parts of the cell content, while leaving the cell itself with the original style. This can lead to messy results, particularly when printing.

Incidentally, Mathematica has spell checking built in. Look under the Edit menu. Please feel free to point out any misspellings you might find in our documentation.

4.6 Printing

Mathematica is a great package in which to develop mathematical solutions. However, ultimately you may need to print a report from Mathematica. Using the Format | Screen Environment menu, the notebook editing environment can be modified to represent the content in different ways. I guess most importantly is to see how your notebook will look when it is printed. To this end Format | Screen Environment | Printout is of some use. However, in this mode you cannot see your page breaks, and often page-count is important. To see page breaks, it is simplest to use the File | Printing Settings | Show Page Breaks menu item. This will switch the screen environment to Printout and show the page breaks. It is not a great environment in which to work, so you can switch back by repeating those menu choices and unchecking the Show Page Breaks option.

Another important point when producing PDF’s from Mathematica, is that Mathematica’s built in “Save As” PDF option tends to give enormous PDF output, particularly if your notebook contains 3D plots. I believe it is rendering every part of the 3D plot rather than flattening it to only what is visible. Anyhow, a better option is to print to PDF using a PDF printer. We have one installed in the PC classrooms.

4.7 Saving

As you build your Mathematica notebook, and experiment with adding snippets of code, getting everything to work just right, the notebook becomes quite a precious item. With a lot of effort spent, it would be unfortunate if the notebook was lost or corrupted. With this in mind you should make sure that you regularly save your notebook, occasionally doing a “Save As” to ensure you have more than one back-up copy. You might also consider using the Cell | Delete All Output menu option to remove all of Mathematica’s kernel returns, making the notebook essentially a text document and less prone to corruption.
In writing your notebook you are programming, and programming can be volatile (fun though!). If your notebook suffers a corruption then there are tools in place to try to help. The expressions:

```mathematica
<<AuthorTools; (*load author tools package*)
NotebookRestore[] (*iterate cells to recover what’s recoverable*)
```

can be used to try to recover corrupt notebooks. The lab demonstrators will happily help in this regard.

### 4.8 Expressions

In common with other programming languages, Mathematica uses a standard input format. Type in and evaluate these expressions:

- `5*2-8/2+6^2`
- `1/3 2`
- `1/3*2`
- `1/(3*2)`

Be careful of the difference between the last three expression above. Make sure you use brackets as required.

In mathematical expressions you should only ever use round () brackets. Square brackets and curly brackets have very specific use in Mathematica as we’ll see later. Notice that in the second expression above the space implicitly defines a multiplication. The standard input method harks back to the BODMAS/PEDMAS calculation precedence sequence that you were likely taught many years ago. This order of operations is strictly adhered to by Mathematica.

### 4.9 More than a calculator

We now want to get Mathematica to work with variables and functions. Let’s first start by setting a value of a variable. Evaluate:

```mathematica
x=5
```

The kernel now has this value for x stored and we can use it later. You can check values using the `Information` function in Mathematica. A short-form for this function is the question mark. Evaluate:

```mathematica
?x
```

You will see that x is defined in the Global context, i.e. accessible in all notebooks, and has the value 5.

### 4.10 Clear or Unset

As you evaluate expressions the kernel fills with values. Sometimes, if a variable has a value in the kernel then it can cause problems. You might want to clear the values in the kernel. You can do this using the `Clear` function, e.g. `Clear[x]`, or we can unset the value, e.g. `x=`. (that’s “x equals full-stop”).

**Exercises 1**

3. Set the values `p=1.5` and `x=6`. Then have Mathematica evaluate `3 x`, `x^3`, `x/p` and `x^3p`.

4. Further down the notebook evaluate `3 z`. The `z` will appear blue and Mathematica will simply return `3z` as it has no known value for `z` to use. It is useful to watch the colours in Mathematica. Colours are used to give you information. In the default colour scheme, blue indicates something that Mathematica does not know about yet, black tells you it is something Mathematica does know about, and purple warns about unbalanced input, e.g. brackets.

5. Correct the “problem” by assigning a value of 5 to `z`, and go back and re-evaluate `3 z`. You don’t need to re-type 3 z, simply use the arrow keys, or mouse, to navigate the insertion point back to the cell containing 3 z, then press `SHIFT+ENTER`.

6. Set `a=1` and `b=5`. Evaluate a `SPACE` b i.e. type a space character between `a` and `b`. Now evaluate `ab`, with the characters together, no space between.
Notice that if you put a space between two expressions, Mathematica assumes these are to be multiplied, so 3 x means 3*x. However, if you type two symbols right next to each other they create a new symbol rather than multiplying the original symbols.

It is important to remember that Mathematica is case sensitive, so capital X is different from lower-case x.

4.11 Building functions

Now that we know how to define variables, we will move on to see how to define functions. This is done in a similar manner to defining variables. However, we need to be aware that Mathematica has an advanced feature which allows it to work with patterns. We will not make immediate use of this feature immediately, but we need to acknowledge its presence, and will look a little closer at function patterns in a later session.

In the meantime, evaluate:

\[ f[x_] := x^2 \]

Note these, very important, things:

- The square brackets are used to enclose arguments of a function.
- The underscore character, after the \( x \) on the left of the expression, is the pattern to which I referred above. \( x_ \) is read as "any expression, to be referred to as \( x \)". It is a dummy variable which could have been called anything, i.e.

\[ f[y_] := y^2 \]

is equivalent. The key thing is to remember the underscore in the definition of functions. It always appears on the left-hand side of function definitions, but never on the right-hand side.

- There is a colon prior to the equals sign in this expression. This colon turns an immediate assignment (as we saw in \( x=5 \) above) into a delayed assignment. The difference being that with an immediate assignment Mathematica will evaluate the right-hand-side of the assignment prior to setting the value. Though in \( x=5 \) the RHS does not need much evaluation! In a delayed assignment, the RHS is not evaluated until the expression is used. What this means is that we can set up the function \( f[x] \) and the RHS is not evaluated until we try to evaluate the function with an argument, e.g. \( f[2] \). It is generally a good idea to set up function assignments as a delayed evaluation, unless you know that the function will be evaluated a huge number of times, in which case you set it up as an immediate assignment, but watch carefully the values assignment to any of the RHS variable is as the time of the function definition.

You have now defined the function and can verify what Mathematica knows about it by evaluating:

\[ ?f \]

Having built the function, the next thing to do is to actually use it. You can simply call the function with the required argument and Mathematica will evaluate the function with that argument.

Exercises 2

7. Evaluate \( f[4], f[1.6] \) and \( f[\Pi] \) - notice there is no underscore necessary in the evaluation - only in the definition.

8. You can also define a variable and plug this into the function, e.g. \( u=6; f[u] \)

9. You can build other functions based on your function: \( g[x_] := f[x]/\sin[x] \)

10. Try building the following functions and evaluating \( f1(2), f2(3) \) and \( f3(9) \):
\[f_1(x) = (x+6)(x+1)\]
\[f_2(y) = (y^3 + 3)/5\]
\[f_3(s) = s^{1/2}/3\]

Answers should be \(f_1(2) = 24\), \(f_2(3) = 6\) and \(f_3(9) = 1\). If you have something else then check your functions definitions (i.e. \(?f_1\), etc.), if you still can’t resolve the issue then ask one of your colleagues, or a demonstrator, to look over your expressions.

### 4.12 Numerical value - the N function

Notice that when you evaluated \(f[\pi]\) in the exercise above, Mathematica returned \(\pi^2\). Mathematica likes to keep everything in as high-precision as possible; Mathematica can calculate \(\pi\) to infinite precision - given time! However, often we want to see a numerical value. To get the numerical value of an expression you can use the \(N\) function. Evaluate:

\[N[\pi, 50]\]

This will show \(\pi\) to 50 digits of precision. You can try more digits if you like - it just takes time to evaluate.

Now, if you are like me, you would have taken the preceding statement as a challenge and probably asked Mathematica to evaluate \(\pi\) to a huge number of digits of precision. Mathematica will be working away and you will see the right-hand cell brace is emboldened as Mathematica works. If it is taking too long then use the \(\text{ALT}+\text{full-stop}\) key-combination to abort evaluation (see this option in the Evaluation menu).

In general, Mathematica will work to the precision of the input expression. Sometimes your code speed can be increased markedly by allowing Mathematica to work at a reduced precision, rather than with exact expressions, e.g. integers.

### 4.13 Palettes and other functions

You are of course not limited to making your own functions, Mathematica has a gamut of built-in functions. As well as those you may immediately think of as being useful, e.g. \(\text{Sin}, \text{Cos}, \text{Exp}\), etc. you will find countless others in Mathematica’s palettes. Use the Palettes menu and have a look at the “Basic Math Assistant” palette. Note that functions all start with a capital letter, and each new word in a function name is also capitalised. You can find a complete alphabetical list of Mathematica’s built-in functions by choosing “Wolfram Documentation” from the Help menu, and typing “Alphabetical Listing” into the search box.

Since Mathematica always starts variables and functions with a capital letter, it is a good idea for you to always use lowercase for your variables. In that way you are avoiding the possibility of a conflict.

### 4.14 Greek alphabet symbols

In physics and astronomy, we often want to use Greek symbols. In the “Basic Math Assistant” palette you will see a “Typesetting” section. Within this section there is a tab to give access to the Greek characters. As some characters are frequently used, I’m thinking primarily of \(\pi\), then it is useful to know shortcuts to these characters. In the “Basic Math Assistant” palette, if you use the mouse to point at the character, then pause for a second or two, you’ll get a pop-up that tells you the shortcut. These generally consist of pressing the \(\text{ESC}\) key, then a letter or two, and the \(\text{ESC}\) key again, e.g. \(\text{ESC} p \text{ESC}\) gives \(\pi\), \(\text{ESC} \alpha \text{ESC}\) gives \(\alpha\), etc. You’ll learn these as you use them, or you can always search the palettes as required.

An alternative means of adding these symbols to your notebook is to use the symbol name, with the first character capitalised, enclosed in square brackets and preceded by a back-slash, i.e. \(\backslash [\text{Theta}]\) gives \(\theta\), \(\backslash [\text{Alpha}]\) gives \(\alpha\), etc.

### 4.15 The semicolon (;)

As you evaluate expressions, Mathematica tends to give you an echo from each expression that you enter, or the result of an evaluation. This can be reassuring in that you see that it is doing what you expect.
However, sometimes it’s useful to be able to suppress this echo. You can put a semicolon at the end of the expression to prevent an output being shown;

Technically, the semicolon is a way of building compound expressions i.e. running Mathematica expression together expr1; expr2; ... Mathematica evaluates all the expressions and gives the output from the last expression, assuming the last expression is not also followed by a semicolon!

4.16 Comments in Input cells

You might want to mark-up your blocks of code - adding comments to remind you of what expressions do. You can add these using an open-bracket-asterisk, then type the comment, and close with an asterisk-close-bracket, i.e. (* This makes a comment in a maths input cell *).

4.17 Lists

So far we have concentrated on evaluating our functions at particular values, but generally we like to see the value of the function over a series of points. Mathematica uses an structure called a List to hold a number of elements. There are a multitude of ways of creating a list, but the two most common methods are using the Range or Table functions, with Table being crucially important.

Evaluate the following two expressions:

\[
\begin{align*}
\text{x1} &= \text{Range}[10] \\
\text{x2} &= \text{Table}[i, \{i, 1, 11, 2\}]
\end{align*}
\]

Use Mathematica’s Help to make sure you understand how Range and Table work.

Notice that in generating a list we see Mathematica’s use of curly brackets. Curly brackets are used to enclose a list of elements. Mathematica uses lists for many reasons. Indeed you can see that in the definition of x2 above, we have a list of four elements, \(\{i, 1, 11, 2\}\). The Table function uses these to define an iterating variable \(i\), a start value 1, end value 11 and step size 2 for the iteration.

Range and Table do very similar things. However, Range can only produce a 1D list, whereas Table can do more complicated list building. There are other functions that also allow you to create lists. These may look closer to methods you might have seen in other programming languages. Evaluate the following four expressions:

\[
\begin{align*}
\text{xido} &= \{\}; \text{Do[AppendTo[xido, i], \{i, 1, 10000, 2\}];} \\
\text{xiwhile} &= \{\}; \text{\textbf{f}i = 1; While[i <= 10000, AppendTo[xiwhile, i]; i = i + 2];} \\
\text{xffor} &= \{\}; \text{\textbf{f}i = 1, i <= 10000, i = i + 2, AppendTo[xffor, i]];} \\
\text{xitable} &= \text{Table}[i, \{i, 1, 10000, 2\}];
\end{align*}
\]

Each of these lines build a list of the odd numbers between 1 and 10000. You should check the lists are identical using the Equal function (\(==\)). Note that this is distinct from the Set function (\(=\)) which is used to assign values.

You can use any of these programming styles to create lists. However, if you wrap each line in the Timing function you can see that Table works most efficiently in Mathematica.

Note that the first argument in the Table function needn’t be a simple expression, indeed it can be a list. Evaluate:

\[
\text{myList} = \text{Table}[\{i, i^2\}, \{i, 1, 10\}]
\]

Your return will be a two-dimensional list, i.e. a list of lists. You might like to think of this as a matrix.

Looking at the output of the above expression, note the number and position of opening and closing curly brackets that are being used to build this two-dimensional list.

You can use the TableForm function to see the variable myList in a more traditional tabular style.

\[
\text{TableForm[myList]}
\]
Sometimes it is quick and convenient to use a function, such as TableForm, with Postfix, i.e.

```
myList // TableForm
```

This might initially seems pointless, but it can save you searching for the start and end of an expression to which you want to apply a function.

**Exercises 3**

11. Use the `Range` function to build the list \{1, 3, 5, 7, 9, 11\}
12. Use the `Prime` function inside a `Table` function to build a list of the first 10 prime numbers.
13. Define a function

```
invest[n_] := 1000*1.05^n,
```

and use `Range` to create a list called “years” that contains the numbers 1 to 5. Evaluate:

```
invest[years].
```

You should now have created a list showing the compound investment return at a rate of 5% p.a. on 1000 sesterces over the next five years. That’s Roman banking for you!

14. Apply the `invest` function as a postfix to `years`.
15. Write an expression to build a list containing a sub-list each of which has the numbers 0 to 2 in steps of \(\pi/4\) along with the sine of the value. The list should look like the following:

```
\{(0, 0), \{\pi/4, 1\}, \{\pi, 0\}, \{3\pi/4, -1\}, \{\pi, 0\}\}
```

**Hint:** You likely want to use the `Table` function, with \(\{x, \sin(x)\}\) as the expression.
16. Now have Mathematica show us the numerical values in the list i.e. no surds or \(\pi\)'s!

### 4.18 List manipulation

We now know how to create a list, but how do we access elements of the list? For this you can use the `Part` function. The short-form for the `Part` function is `[[ ]]`. Evaluate:

```
myList=Table[Fibonacci[n],\{n,1,50\}]
myList[[10]]
```

This builds a list of the first 50 Fibonacci numbers and assigns this to the symbol `myList`. It then pulls out the tenth item in `myList`. Of course we could have just evaluated the function `Fibonacci[10]` to get the same result, but that’s beside the point!

You can check the size of lists using either `Length` (for 1D lists) or `Dimensions` for higher dimensional lists. Type the following to verify that you have 50 items in `myList`:

```
Length[myList]
Dimensions[myList]
```

You can select items from a list based on defined criteria. Try the expression `Select[myList, PrimeQ]` to return all the prime numbers from your list of Fibonacci numbers.

**Exercises 4**

17. Define two lists, a and b, with values \{1, 2, 3, 4, 5\} and \{1, 4, 9, 16, 25\} respectively. You could type these, but best to try to figure out how to use `Range` or `Table` to generate them automatically. If I see you having typed them in then I will ask you to augment the list to the first 100 squares. You won’t be so keen to type them in then!
18. Evaluate separately:

```
Join[a, b]
Union[a, b]
Intersection[a, b]
```

Note the action of these three functions.
19. Define \( c \) and \( d \) as being \{a, 6, 7\} and \{b, 36, 49\}, respectively. Notice that there are extra, undesirable, brackets. Define \( e=\text{Flatten}[c] \) and \( f=\text{Flatten}[d] \) to remove all the inner brackets, then define \( g=\{e, f\} \) to return the complete 2D list \{\{1,2,3,4,5,6,7\},\{1,4,9,16,25,36,49\}\}.

20. The result from the step above is very nearly a useful structure. However, we’d prefer to have each value in the first list associated with its square in the second list. Use the \textit{Transpose} function to rearrange the list, and the \textit{TableForm} function to ultimately show the data in tabular form:

```
Alphabetical list of functions met so far.
explicitlyUsed = \{\text{AppendTo}, \text{Clear}, \text{Dimensions}, \text{Do}, \text{Fibonacci}, \text{Flatten}, \text{For}, \text{Intersection}, \text{Join}, \text{Length}, \text{N}, \text{Part}, \text{Postfix}, \text{Prime}, \text{PrimeQ}, \text{Range}, \text{Select}, \text{Sin}, \text{Table}, \text{TableForm}, \text{Timing}, \text{Transpose}, \text{Union}, \text{While}\};
implicitlyUsed = \{\text{Divide}, \text{Equal}, \text{Information}, \text{List}, \text{Plus}, \text{Power}, \text{Set}, \text{Sqrt}, \text{Subtract}, \text{Times}, \text{Unset}\};
```

So far we have discussed lists, how to manipulate them and display them in tabular form using the \textit{TableForm} function. Scientists like only one thing better than a table for displaying results and that is a graph . . .

### 4.19 Plotting graphs

One of the best ways, if not \textit{the} best way, to assess trends and spot anomalies in results is to plot them on a graph.

Plotting functions, or data in the form of lists, is very straightforward in Mathematica.

#### Plotting using a function

To create a 2D plot of a function you can use the \textit{Plot} function. For example, to plot the Sinc function, i.e. \( \frac{\sin(x)}{x} \), you use:

```
Plot[Sinc[x], \{x,-10,10\}]
```

The \textit{Plot} function has two primary arguments:

- The first argument is the function that we want to plot. If you want to plot multiple functions on the same graph, then this first argument can be a list of functions e.g. \{\( \cos(x) \), \( \sin(x) \), \( \tan(x) \)\}.
- The second argument is a list defining the independent variable and its range i.e. \{\( x \), \( x_{min} \), \( x_{max} \)\}. Notice that you do not have an option here to specify a step-size in the range, Mathematica takes care of this, specifying a default, but adaptable, number of points to use in plotting. You can of course change this plot \textit{option}, and this is what we’ll talk about next.

#### Options

Although the \textit{Plot} function has two primary arguments, Mathematica gives you a slew of options that you can apply to your plot to make it look exactly the way you want it to look. These options are called, well,
Options! You can find out about all the available options for a function using the Options function, e.g. Options[Plot] will give you a list of all the options for Plot and will show you their default values. To set an option you have to use Mathematica's rule notation. We'll talk more about rules later, but suffice it to say for the moment, you set the option using an arrow (\[Rule\]). You can get this from the palettes, but it's quickest just to type a minus sign immediately followed by a greater-than sign, i.e. \(-\rightarrow\). Mathematica will interpret this as the rule arrow.

So to actually set an option, let's say we want to change the colour and thickness of the sinc line in our plot to red and thick, we can write:

\[
\text{Plot[Sinc[x],\{x,-10,10\},PlotStyle\rightarrow\{Red,Thick\}]}
\]

Notice that we've introduced a third, optional, argument to the function. Red and Thick are built-in constants representing values RGBColor[1,0,0] and Thickness[0.01] respectively.

When plotting a graph, it is always good practice to give it a title and axes labels. Use of the Frame and FrameLabel options give a satisfactory looking output:

\[
\text{Plot[Sinc[x],\{x,-10,10\},PlotStyle\rightarrow\{Red,Thick\},}
\]

\[
\text{PlotLabel\rightarrow"The Sinc function", Frame\rightarrow True, FrameLabel\rightarrow\{"x","Sinc[x]\}\]}
\]

Throughout the module, whenever you make a plot you are expected to label the axes and give the plot a title.

You might also want to draw gridlines to make identifying values on the graph easy. Try adding the GridLines->Automatic plot option.

Exercises 5

1. Now that you have plotted a thick, red sinc curve, make a plot containing both a thick, red sinc curve, and a thick, blue sinc^2 curve. Note that Mathematica does not follow the usual trigonometric power convention. So, for example,

\[
\text{Sin[x]^2}
\]

is used instead of \(Sin^2[x]\]!

2. Investigate the use of the Tooltip function to allow Mathematica to give the user a pop-up label when they point the mouse at the plotted curves.

3. Change the GridLines plot option in such a way that we have grid lines spaced one unit apart on the x axis, and from -0.25 to 1.0 in steps of 0.25 on the y axis.

4. Make a copy of the Plot statement and change the GridLines option such that you have one horizontal and one vertical grid line. These should cross at the right-hand peak i.e. (7.7, 0.13).

Plotting using a list

So if you thought that plotting functions was pretty straightforward, plotting lists is even easier. The first thing we need is a list...

Let's imagine we are doing a laser beam profiling experiment, and expect a nice Gaussian profile of the form \(ae^{-bx^2}\). However, as with any experiment, we have errors in our experimental data. The following expression will generate a list of intensities with a Gaussian profile, but with some random variation provided by the RandomReal function.

\[
\text{myProfile = Table[RandomReal[\{0.8, 1.2\}] Exp[-0.1 x^2], \{x, -10, 10, 0.5\]}
\]

Now that we have the list, we can simply plot this list using the ListPlot function, i.e.

\[
\text{ListPlot[myProfile, Frame \rightarrow True,}
\]

21
PlotLabel -> "Measured Gaussian intensity profile",
FrameLabel -> {{"x (arb. units)"}, {"Intensity (arb. units)"}}]

Note: You should be forensically examining the various expressions that you are coming across. Dissect them and make sure you understand how they work. Before moving on, make sure you know what Table, RandomReal and Exp are doing in this expression. If you are uncertain then ask a demonstrator at the PC classroom session.

Exercises 6

5. Since we only gave Mathematica a 1D list of intensities, it uses the list-position as the x-value in the plot. It might be nice to have the actual x value that we used when calculating the list, to give us "proper" coordinate pairs. Modify the definition of the myProfile list to include the x-value alongside the intensity. Re-plot the new 2D list, allowing it to show a corrected x-axis scale. If you’re not sure how to do this then re-read the “Ranges and Lists” section from session one.

6. Investigate the ListPlot options to find out how to join the points on this plot.

7. There is an alternate list-plotting function that always joins the dot, what’s it called? How are you going to find it? \[F1\] on ListPlot then "See Also".

8. The chances are that your join-the-dots plot is not going to be very satisfactory. Ideally we’d like to fit a nice, smooth curve to the plot. If we know that the underlying function is Gaussian of the form \[ae^{-bx^2}\], we can use the FindFit function to determine the values of a & b that fit our data points. Since we used a and b previously, it would be a good idea to unset these before using a & b as parameters.

4.20 Rules

Having successfully used the FindFit function you will end up with a list something like:

\[\{a \to 0.989217, b \to 0.0909006\}\]

This is a list of rules. Mathematica has not set the values of a or b, it is telling us that these are the values that solve the problem. We then need to use the values. The most convenient way to use the value is to substitute them into an expression. Try evaluating:

\[a \text{ Exp}\left[-b x^2\right] /. \{a \to 0.989217, b \to 0.0909006\}\]

Now we have an expression that we can define as a function, and use like a regular function. Normally you would set variables to the result of functions to make the results reuse easier. So our block of code to get a function that fits our data could be:

\[\text{model} = a \text{ Exp}\left[-b x^2\right];\]
\[\text{sol} = \text{FindFit}[\text{myProfile}, \text{model}, \{a,b\},x]\]
\[\text{myFit}[x_] := \text{model}/.\text{sol}\]

myFit would then be a function that we can plot using the Plot function. Notice that we have defined an expression for the model. As we use the Gaussian expression twice this makes good sense.

This method of solving as a list of rules, then using these rules, is commonplace in Mathematica. In the definition of myFit[x] above we are using a function called ReplaceAll (short-form /.). It would be useful to read the Help file to make sure you understand how ReplaceAll works. We will use it a lot throughout the module, so you should make sure you understand it now.
Exercises 7

9. Plot the fitted Gaussian function with your values of a & b determined from FindFit.
10. Investigate the Show function and find out how to overlay the two plots to see the data (myProfile) and the fit (myFit) on the same graph.
11. An alternative, and I think more elegant method of putting data and a fitted line together, is to use the Plot function to draw the fitted function, and to use a combination of Epilog and Point. Plot the fitted function, and investigate the use of the plot option Epilog->{Point[myProfile]}.
12. Now try extending the range on the x-axis such that \(-100 \leq x \leq 100\). Depending on the version of Mathematica, this might introduce a warning from Mathematica that some of our values, on the wings of the Gaussian, are very small. As we are not doing any further calculations with these numbers we can ignore the warning. Indeed, if we want to hide this type of underflow warning we can use the command Off[General::munfl].
13. The new plot does not look like a Gaussian. This is because Mathematica zooms to show variations. To see the whole function use the plot option PlotRange. Use the help file to find out the details.

4.21 Manipulate and animate

I hope you would now agree that Mathematica can easily produce nice graphs. It can sometimes be useful to interactively see the effect of a parameter on a graph. For this we can use the Manipulate or Animate functions. These very closely mirror each other, so I’ll concentrate on Manipulate and leave you to investigate Animate yourself.

To see how Manipulate works we’ll use a modified version of the Manipulate Help entry to show a travelling wave. Type the following into Mathematica:

Manipulate[Plot[Sin[5t-ϕ],{t,0,2π},AxesLabel->"t", "Sin[5t-ϕ]"],{ϕ,0,2π}]

This allows you to interactively change ϕ (between 0 to 2π) and to plot \(sin(5t - \phi)\) between \(0 \leq x \leq 2\pi\) for each new value of ϕ. You can use the slider to change the value of ϕ. You can also display additional controls using the small + sign at the right of the slider. Here you can use the play button to automatically change ϕ. In this way Manipulate even more closely resembles the Animate function.

Exercises 8

14. In the example above we have effectively set \(\omega = 5\). Change the Manipulate expression to allow the user to change both \(\phi\) and \(\omega\), with \(1 \leq \omega \leq 10\). See if you can have Mathematica set the initial value of \(\omega\) to be 5 when the Manipulate expression is evaluated.

4.22 Scope of variables

Let’s look at manipulating the Sin curve again, but this time we will take the Sin expression on to its own line and give it a name, then we will use that name in the Manipulate expression.

sincurve = Sin[5t-ϕ];
Manipulate[Plot[sincurve,{t,0,2π},AxesLabel->"t", "Sin[5t-ϕ]"],{ϕ,0,2π}]

When you evaluate these expressions, you end up with an empty graph that shows no variation as \(\phi\) is changed. The reason for this is that the symbol \(\phi\) used in the Manipulate function is local to that function, and when sincurve appears with its own global version of the \(\phi\) symbol, they are treated as being distinct. So a value for \(\phi\) is never substituted into sincurve, and so it cannot evaluate to a number that can be plotted - hence the empty graph.

A convenient means of circumventing this issue is to define any expressions that you want to manipulate as a function. Try modifying your entry to the following.

sincurve = . (*unset the old definition to prevent an "is Protected" error*)
sincurve[ϕ_] := Sin[5t-ϕ]
Manipulate[Plot[sincurve[ϕ],{t,0,2π},AxesLabel->"t", "Sin[5t-ϕ]"],{ϕ ,0,2π}]

Now, when Manipulate sets a value of \(\phi\), this value is sent to sincurve as an argument, coincidentally also called \(\phi\), and so can be used in its evaluation.

4.23 3D plotting

We have seen how Plot and ListPlot are used to plot functions and lists, respectively, in 2D. In 3D there are similar functions called, unsurprisingly, Plot3D and ListPlot3D. I’ll leave you to look up help on ListPlot3D to find out how it works. It’s fairly intuitive. There are some nice “neat examples” if you scroll down the help page on the ListPlot3D function.

Exercises 9

Let’s look at using Plot3D. Using this we can plot a nice “egg-box” function using the following expression:

\[
\text{Plot3D}[\sin(x)\cos(y),\{x,-2\pi,2\pi\},\{y,-2\pi,2\pi\},\text{PlotStyle}\rightarrow\text{Opacity}[0.8],
\text{AxesLabel}\rightarrow\text{"x", "y"},\text{PlotLabel}\rightarrow\text{"Egg box function"}]
\]

We can rotate the plot by clicking and dragging with the mouse to change the viewing angle. Notice that we have also set the PlotStyle to make the surface partially transparent.

Let’s take a step back to 2D plotting for a moment. The intensity of the diffraction pattern from a circular aperture is given by the formula:

\[
I \propto \left( \frac{J_1(ka \sin \theta)}{ka \sin \theta} \right)^2,
\]

where \(a\) is the aperture radius, \(k = \frac{2\pi}{\lambda}\), \(\lambda\) is the wavelength of the illuminating light, and \(\theta\) is the far-field angle between the normal to the aperture and the direction of propagation of the light. If you think about the physics of this experiment, you should be able to choose realistic values for \(\lambda\) & \(a\), and a sensible range of values for \(\theta\) when plotting.

\(J_1\) is the “Bessel function of the first kind and the first order” and is accessed using the BesselJ function in Mathematica, i.e.

\[
\text{int}[\theta] := (\text{BesselJ}[1, k a \sin[\theta]]/(k a \sin[\theta]))^2.
\]

Notice that the function is called int and we have not used capital i as the function name. This is for good reason, any suggestions why?

15. Make a plot of intensity vs. \(\theta\) for this 1D case.

16. Define a function that returns the angle \(\theta\) as a function of the position \(x\), the distance from the central maximum on a screen one metre (\(R\)) from the aperture.

17. Use the function defined above to re-plot the intensity, but now against \(x\).
18. Use the function \texttt{FindMinimum} to determine the distance of the first minimum from the central maximum.

19. To make a 3D plot of the Airy diffraction pattern intensity for the circular aperture, define a function that returns the angle $\theta$ as a function of the position $x$ and $y$, where $x$ and $y$ are the Cartesian distances from the central maximum on the screen.

20. Use the function defined above to make a 3D plot of the intensity as a function of the positions $x$ and $y$.

21. If the default option in your 3D plot make the plot look a little ragged then you should increase the number of plot points. You might also investigate how to change the colour, mesh, etc.

22. It might be nice to see a top-down projection of this pattern. Investigate the \texttt{DensityPlot} function to see if you can get a picture of the central spot and the fringes as you might see them if viewed on the screen.

\textbf{Alphabetical list of additional functions met}

\texttt{explicitlyUsed = \{Animate, BesselJ, Cos, DensityPlot, ErrorBar, ErrorListPlot, Exp, FindFit, FindMinimum, ListLinePlot, ListPlot, ListPlot3D, Manipulate, Off, Opacity, Options, Plot, Plot3D, RandomReal, ReplaceAll, RGBColor, Show, Sinc, Tan, Thickness, Tooltip\};}
5 Mathematica introduction - PC classroom sessions - Week 2

By now you should be getting moderately familiar with the Mathematica interface, be confident with lists and how to create them, and know how to generate plots. In this session we will get Mathematica doing a whole load of more maths-like things ...

5.1 Differentiation

Mathematica can handle differentiation and has a function for both total differentiation, \texttt{Dt}, and partial differentiation, \texttt{D}. In general we can happily use partial differentiation as this assumes that all quantities that do not explicitly depend on the variables given are taken to have zero partial derivative. Try evaluating this block of code (put them all in one cell):

\begin{verbatim}
  f[x_] := Exp[-x^2];
  f'[x] (*first derivative, prime notation*)
  f''[x] (*second derivative, prime notation*)
  D[f[x],{x,5}] (*fifth derivative, using the D function*)
\end{verbatim}

For functions of more than one variable, try evaluating:

\begin{verbatim}
  f[x_,y_] := Exp[-x^2+x*y];
  D[f[x,y], x] (*first derivative with respect to x, fixed y*)
  D[f[x,y],{y,2}] (*second derivative w.r.t. y, fixed x*)
  D[f[x,y],{x,2},{y,1}] (*second derivative w.r.t. x, first derivative w.r.t. y*)
\end{verbatim}

Use functions \texttt{Simplify} and \texttt{FullSimplify} to simplify the last expression.

Note that the prime notation for derivatives is only applicable to functions of one variable, else Mathematica would not know to which variable the prime applied.

5.2 Integration

You can find more traditional ways in which to write differentiation in the calculus section under "Basic Commands" in the "Basic Math Assistant" palette.

You can do both definite and indefinite integration using Mathematica. These are performed using the same function, \texttt{Integrate}, but differ in whether or not limits of integration are supplied to the function. For example, evaluate the following:

\begin{verbatim}
  Integrate[Sec[x],x]
  Integrate[Sec[x],{x,0,1}]
\end{verbatim}

In the former we are making an indefinite integral, whereas in the latter we have supplied limits to the variable of integration and so get a definite integral. The latter gives a very precise answer. Do you recall how to make it appear as a number?

There are, at least, two means of making an expression in Mathematica give you a numerical answer.

- We have seen the use of the \texttt{N} function for this purpose earlier, so we could simply have typed:
  \texttt{N[Integrate[Sec[x],{x,0,1}]]}. Incidentally, Mathematica also has a function \texttt{NIntegrate}, this uses numerical methods to perform the integration - it can be useful if an algebraic integration cannot be performed. \texttt{NIntegrate} will implicitly give you a numerical answer, though it may not be as accurate as \texttt{N[Integrate[...]]}. A numerically derived answer will in general be faster. You can verify this using the \texttt{Timing} function.
Earlier we mentioned that Mathematica likes to work to high precision when it can. In the second integral we have defined the limits as integers. Integers are infinitely precise, they do not suffer from any rounding errors as is the case with numbers held in floating-point format. Since the input has this high (infinite) precision, Mathematica gives us high precision output, i.e. \(2 \, \text{ArcTanh}[\tan \left(\frac{1}{2}\right)]\). Reducing the precision of the input forces Mathematica to give us a reduced precision output, e.g. \(\text{Integrate}[\sec(x), \{x, 0, 1.\}]\). Notice that all I have done is add a decimal point to the upper integration limit. This then changes this number from an integer into a floating-point number (a real number). This floating-point number is not infinitely precise, so Mathematica defaults to a lower precision (MachinePrecision) and hence gives us a numerical answer. As with life in general, it is the case that the answer will reflect the precision of the question.

5.3 Piecewise functions

Sometimes we come across functions which are not smooth but are piecewise continuous, e.g. when it is necessary to define boundaries or junctions between functions in physics where we have a physical boundary. Mathematica has three primary means of bolting together functions in this manner:

- The Piecewise function:

  \[
  \text{Plot[Piecewise[}\{\{x^2, x<0\},\{0, x<2\},\{10-4x, x>=2\}\},\{x,-2,3\}]
  \]

- The If function:

  \[
  \text{Plot[If[x<0, x^2, If[x<2, x, 10-4x]],\{x,-2,3\}]
  \]

- The Which function:

  \[
  \text{Plot[Which[x<0, x^2, 0<x<2, x, x>=2, 10-4x],\{x,-2,3\}]
  \]

These all produce the same plot so you can use whichever makes most sense to you. That said, I’d tend to steer clear of using the If function, unless for a very simply purpose, as it requires nesting (one If inside another) and can get unwieldy to write and bug-check.

5.4 Differential equation solving

An inordinate amount of physics comes down to solving differential equations. When we try to model or analyse a physical system, we tend to use mathematics to write equations governing the system. These equations are invariably differential. Mathematica can solve differential equations either algebraically (symbolically) using \texttt{DSolve}, or numerically using \texttt{NDSolve}. However, typically we will not get a system of equations that can be solved algebraically, so likely we will default to using the \texttt{NDSolve} function to solve our differential equations numerically.

To solve a differential equation numerically, we not only need the differential equation, but we also need \textit{initial conditions}. The rule is that for every order and every variable of the differential equations you need one initial condition, i.e. if you have a second order differential equation in two variables, then you will need 4 initial conditions.

Before going on to evaluate the following block of code, note that when we build \textit{equations} in Mathematica we use the double equals sign. This is distinctly different to the use of the single equals sign. Double equals, \(==\), defines the left-hand-side to be equal to the right-hand-side, whereas the single equals or colon-equal, \(=\) or \(:=\), means to set the variable or function on the LHS to the value on the RHS.

Now, try entering and evaluating the following lines of code. I have separated the \texttt{NDSolve} function onto separate lines to improve readability. You are at liberty to replicate this style, or to string the whole function along one line, noting that Mathematica will line-wrap as necessary.
\[ \theta = \pi / 6; v_0 = 10; \]

\[ \text{eqs} = \{ y''[t] == -9.8, x''[t] == 0 \}; \]

\[ \text{inits} = \{ x'[0] == v_0 \cos[\theta], y'[0] == v_0 \sin[\theta], x[0] == 0, y[0] == 0 \}; \]

\[ \text{sol} = \text{NDSolve}[\{ \text{eqs}, \text{inits} \}, \{ x, y \}, \{ t, 0, 10 \}][[1]]; \]

\[ \text{ParametricPlot}[\{ x[t], y[t] \} /. \text{sol}, \{ t, 0, 1 \}, \]

\[ \text{PlotLabel} \to \text{"Simple trajectory"}, \text{AxesLabel} \to \{ \text{"x"}, \text{"y"} \}]; \]

It's worthwhile noting:

- This is simple projectile motion; we are solving for \( x \) & \( y \), over a time interval, \( \{ t, 0, 10 \} \). It involves second order differential equations in two variables, hence we have four initial conditions:

\[
\begin{align*}
x'[0] &= v_0 \cos[\theta] \\
y'[0] &= v_0 \sin[\theta] \\
x[0] &= 0 \\
y[0] &= 0
\end{align*}
\]

- A quick way to find the extent of a function is to triple-click the function name. The first click places the cursor, the second selects the function name, and the third selects the entirety of the function. Triple-click on the word `NDSolve` in your block of code to check the extent of that function.

- Having triple-clicked `NDSolve` in the step above, you will notice that the function ends just before the end of the line, and immediately thereafter we have `[[1]]`; We'll talk about the semicolon and the `[[1]]` (First part) in the next two items:

- You should know that the semicolon is there to suppress the output from that line - if you didn’t then please re-read the earlier section on the use of semicolons from the first introductory session. If you remove the semicolon and re-evaluate the projectile motion block of code you’ll see something potentially unexpected...

  `NDSolve`, like many other solving functions in Mathematica, does not yield answers directly, but yields a rule. We talked about rules before, so that is not the surprise. The surprise is the right-hand-side expression. Since we performed a numerical solution, Mathematica gives us an `InterpolatingFunction` for our solution. This is simply a look-up table of values, alongside a method of interpolating between these values. The `InterpolatingFunction` behaves exactly like any other function. We then use the solution (variable name `sol`) with the ReplaceAll (`/.`) mechanism that we discussed before. If you’re still not getting the idea of rules and replace-alls, then now is the time to get it right. Confer in your colleagues, and if that is not helping, ask a demonstrator.

- Now try removing the `[[1]]` and re-evaluating the projectile motion block of code. You should see that the solution has an extra set of curly-brackets. When we solve the differential equations, Mathematica is set up to return all the solutions. However, in this case there is only one solution. When we append `[[1]]` to the solution, we are simply taking part one out of the list of solutions. There is only one solution, so effectively we are removing a set of enclosing brackets. This is non-essential here, but having our solution in this slightly abridged format will help us as our solutions are applied in later examples.

- The last line of our code-block contains the `ParametricPlot` function. This allows us to calculate \( x \) and \( y \) using \( t \) as a parameter, and plot the resulting trajectory.

The projectile example shown above is so simplified that we could have in fact used `DSolve` to calculate an algebraic solution. Copy the three lines of code into a new input cell and change the second line to use `DSolve` instead of `NDSolve`. Make sure you have removed the semicolon so you can see the algebraic solution.

**Note:** `NDSolve` solves for a predefined range of values of \( t \), whereas `DSolve` makes an expression that is valid for all \( t \). This means that the structure of the `NDSolve` and `DSolve` functions are a little different; though it seems that including a range for \( t \) in `DSolve` is tolerated, and ignored, by recent versions of Mathematica.

The algebraic solution is nice. However, the numerical method gives us the possibility of adding other factors to the system, e.g. air resistance or Magnus force, that are not solvable algebraically. Hence `NDSolve` tends
to be the more useful for physicists and astronomers, else we end up back at the spherical chicken in a vacuum!

Before leaving this NDSolve projectile example, you might want to ponder why we plotted over a time range from \{t,0,1\}, whereas we solved for \{t,0,10\}. In the next session we will look at how we can stop the numerical integration when particular conditions are met, e.g. when the projectile hits the ground.

5.5 Finding Roots

Finding roots, or intersections, is often useful in physics. For example, let’s find all the intersections of \( y = \sin(x) \) and \( y = (4+x)/8 \). First plot the function to get an idea of where the intersections lie.

\[
y_1[x_] := \sin[x];
y_2[x_] := (4 + x)/8;
Plot[{y_1[x], y_2[x]}, \{x, -20, 20\}, PlotLabel->"Root finding", AxesLabel->{"x","y"}]
\]

So we see 5 intersections all between \(-10 < x < 5\). We can use the FindRoot function to determine these one at a time, looking for a root near a starting value (in this case near \( x = -3 \)):

\[
\text{FindRoot}[y_1[x] - y_2[x], \{x, -3\}]
\]

Incidentally, by default, the FindRoot function uses Newton’s method. You can read about this, and other computational methods that Mathematica uses, in any Numerical Analysis textbook. It is easy to look on Mathematica as a black-box that provides the correct answer. However, you should always apply critical thinking to any results; primarily considering whether they make physical sense.

5.6 Solving systems of equations

The Solve function is a useful and flexible function for solving equations, or systems of equations. Use Solve to determine the values of \( x, y, \) & \( z \) from the following simultaneous equations:

\[
\begin{align*}
2x + 3y - z &= -13 \\
x - 6y + 4z &= 7 \\
-6x - 2y + 2z &= 52
\end{align*}
\]

5.7 The bakery problem

Try setting up and solving this bakery problem - out of context of physics, but a nice challenge I think:

A bakery sells packs of baked goods. One morning they sold enough packs to take £1523.10. The croissants come in packs of 30 and cost £76.60 per pack. The bagels come in packs of 18 and cost £45.20 per pack. The muffins come in packs of 12 and cost £23.10 per pack. In total 642 individual items (i.e. not packs) were sold. Only complete packs were sold. How many packs of each product were sold?

Note that we can build two equations with three unknowns. However, there are two additional pieces of information implicit in the problem:

- They sold only complete packs, so the numbers of pack must be integer.
- They only sell packs, never buy, so the numbers of packs must be greater than or equal to zero.

Set up the two equations and solve them, making use of the additional information. I’d suggest using the Solve function, but if you have an alternative plan, then feel free to solve it your own way. If you need assistance in figuring out how to define that the solution must be positive integers, then take a look at the fourth basic example in Solve’s help entry.

Alphabetical list of functions first met in this session

explicitlyUsed = \{Dt, D, Simplify, FullSimplify, Integrate, Sec, NIntegrate, Piecewise, If, Which, NDSolve, ParametricPlot, DSolve, InterpolatingFunction, FindRoot, Solve, Point\}
In the preceding material we have systematically covered many of the concepts and methods required in order to allow you to start using Mathematica to solve problems in Physics and Astronomy. There are another few themes that you should be aware of. These will be highlighted in this session.

5.8 Using strings

Sometimes there is a need to use and manipulate string variables. Strings are simply a sequence of characters e.g. "this string". We can set a variable to have a string value. Various functions can be performed on strings. Useful functions are StringLength and ToString which respectively return the number of characters (including spaces) in a string, and convert symbols to strings. We can join strings together using the StringJoin function which is mostly used in the short form <>.

```mathematica
str = " this string ";
a = 45;
StringLength[str]
str1 = "abcd" <> str <> "xyz" <> ToString[a]
StringLength[str1]
```

String manipulation can also be used to extract and change parts of strings.

```mathematica
StringTake[str1, {6, 17}]
StringPosition[str1, "this"]
StringReplace[str1, "this" -> "that"]
```

What does the return \{\{6,9\}\} mean from the middle expression above? Note that you may get slightly different numbers depending whether you observed all the spaces in the input strings. Spaces count as characters too!

5.9 Cell formatting

You can use Mathematica a bit like a word processor; but one with great mathematical abilities. You can format cells and build a document structure/hierarchy. Use the NotebookOpen function to open the notebook at the URL:

```mathematica
path = "https://www.st-andrews.ac.uk/physics/ph3080/"
NotebookOpen[path <> "intro/SampleNotebook.nb"]
```

**Important:** The variable `path` will be used with this value in the rest of the module, including homework tasks and the exam, when we import data or open notebooks.

The style and structure is added using the Format | Style menu. You should ideally only structure whole cells. You can do this either prior to creating the cell, or by selecting the cell brace and changing the formatting from the Format | Style menu. Note that things start to get unmanageable if you structure the content of a cell with various cell styles, best to stick to formatting complete cells. You can of course use **Bold**, *Italics*, etc. within a cell as you like.

5.10 Using data files

Data can be read from files using the Import function. Likewise you can write data, from Mathematica to a file using the Export function. I'll leave you to investigate Export by yourself (using \[F1\] of course), and we will look at a couple of Import examples.

**Sudoku**

The sudoku puzzle shown below has a mistake (highlighted), you can see that the line totals are not consistent.
Import the sudoku grid using the following command:

\[
\text{data} = \text{Import[path <> "intro/sudoku.csv","CSV"]}
\]

- Use the Grid function to display the sudoku grid. The Dividers->All option might also be useful.
- Calculate the total for each column using:
  \[
  \text{Total[data]}
  \]
- Calculate the total for each row using:
  \[
  \text{Total[Transpose[data]]}
  \]
- Using the Part function (short-form data[[ ]]), set the erroneous element in data to the correct value to ensure that the column and row totals are consistent. Display the modified value of data in a Grid for comparison with the earlier grid.

**Data import and fitting**

In session two we used the FindFit function to fit a Gaussian to our simulated laser beam profile experiment results. Let's import some data and fit a quadratic:

\[
\text{imp} = \text{Import[path <> "intro/datafile.txt"]}
\]

- This imports the information from the file. However, since the file has a .txt extension, Mathematica imports this as text. To import the information in a usable manner you should use the "Data" element argument in the Import function (the help page on Import has more details).
- The imported list contains two lines of header information. We can extract using the Part and Span functions

\[
\text{data=imp[[3;;-1]]];
\]

\[
\text{model} = \text{a0} + \text{a1 x} + \text{a2 x}^2;
\]

\[
\text{fit} = \text{FindFit[data, model, \{a0, a1, a2\}, x]
\]

\[
\text{Plot}[\text{model /. fit}, \{x, 0, 0.4\}, \text{Epilog} \to \{\text{Red, Point[data]\}}]
\]

**Working with image data**

The following block of code will import an image, extract the pixel data and dimension sizes. It will then calculate the centre of gravity of the intensity in the image and plot the image and a cross identifying the centre of gravity.

\[
\text{img = Import[path <> "intro/CoG_image.jpg","JPG"]}
\]

\[
\text{imgData} = \text{N[ImageData[img]]};(*N turns data to floating point numbers for speed*)
\]
{yMax, xMax} = Dimensions[imgData]
totI = Total[imgData, 2]
xMean = Sum[x imgData[[y, x]], {y, 1, yMax}, {x, 1, xMax}]/totI
yMean = Sum[y imgData[[y, x]], {y, 1, yMax}, {x, 1, xMax}]/totI
Graphics[{Raster[imgData],
{Red, Line[{{1, yMean}, {xMax, yMean}}], Line[{{xMean, 1}, {xMean, yMax}}]}}]

This again is an instance where as well as typing in the code accurately, you should also be dissecting the code to understand how it works. The lab demonstrators will be happy to help you clarify anything of which you are unsure. Incidentally, if you think this result looks a little off-centre, then test the code on the "circle.jpg" image in the same location. Should confirm that all is fine.

5.11 Oil pipeline cost problem

There is a classic analog to Snell’s law that discusses a lifeguard’s dilemma in reaching a swimmer in distress. The dilemma is to determine how far the lifeguard should run on the beach and how far they should swim; running being faster than swimming. To change the context a bit, let’s do exactly the same problem, but looking at it from the point of view of laying an oil pipeline.

We want to determine the distance \( x \), at which the pipeline enters the sea. The value should minimise the overall cost of the pipeline installation. Develop the solution by following these steps:

- Write a function, called \( \text{onLandCost} \), which is a function of \( x \), and reflects the cost of laying the pipeline on-land.
- Write a function, called \( \text{atSeaCost} \), which is a function of \( x \), and reflects the cost of laying the pipeline at sea.
- Use the NMinimize function to minimize the total cost, \( \text{onLandCost}[x] + \text{atSeaCost}[x] \), of the complete pipeline from oil-rig to refinery in terms of \( x \).
- Determine the values of the angles \( \theta_1 \) and \( \theta_2 \).
- Compare the ratio of \( \frac{\sin(\theta_1)}{\sin(\theta_2)} \) to the ratio of the cost per km of pipeline in the sea to that on land.

You should now hopefully be convinced that Snell’s Law (an application of Fermat’s Principle) applies to many differing situations where path length must be optimised.

5.12 Pure functions

Previously, when defining a function we have given the function a name e.g. \( f[x_] := 10 \times \). This is useful for repeated use of the function. However, sometimes we only want to use a function once, and giving it a name seems wasteful. In these instances we can use what Mathematica calls a “pure function”. These are
introduced without giving them a name. e.g. Function\[x, 10 x\] behaves as the definition for \(f\) above. So \(f[3]\) will yield the same answer as Function\[x,10x\][3].

We can even write this pure function in its short form using the ampersand (&). In this notation the function \(f[x]\) above would be written as 10#&. The hash is called a slot and it is where the values are plugged in to the function. The ampersand is the short form for the function Function.

The most powerful use of pure functions comes from their use in processing lists.

\[
\text{Map[If[#>10,10,#]&,}, f 5,3,1,11,7,15,2{]} (*anything above 10 becomes 10 - a threshold function*)
\]

Another common use of pure functions is in setting the colouring in plots. Mathematica's default colour scheme is not always what is desired, so we can use the ColorFunction plot option to modify the scheme. Usually the function is only needed once, so a pure function is ideal.

\[
\text{Plot3D[Sin[x y],}, f x,0,3{],} f y,0,3{],}\text{PlotLabel->"Plot of Sin[x y]","AxesLabel->"x","y"},\text{ColorFunction -> (GrayLevel[#]&)}{]}\]

Depending on what data is being passed, you might also see \#1, \#2 to distinguish between multiple parameters. Try changing the has # in the above expression to \#1, \#2, & #3 to see the impact on the output. What then do \#1, \#2 & \#3 represent in this case?

5.13 Programming with rules and patterns

We have seen rules, e.g. \(x->5\), before and mentioned how the use of certain functions often results in a list of rules for their output. We saw how the ReplaceAll (/. ) function is used to implement the rules. For example, \(2x+y/.\{x->5, y->3\}\) would result in the output 13 as in the expression \(2x+y\) we replace \(x\) with 5 and \(y\) with 3, then \(2 \times 5 + 3\) gives 13.

Rules and patterns can be used together to provide some flexible programming. Consider the pattern \(x\_\text{Integer}\). This defines a pattern, that is given the name \(x\), and refers to the set of integers. Using this pattern we can construct a rule based expression:

\[
\{1,-23,-3.5,4.1,7\}/.x\_\text{Integer}->0
\]

Here, Mathematica moves through our list of numbers element-by-element, and wherever the pattern matches (i.e. where there is an integer) it applies the rule. So all integers become zero.

\[
\{1,-23,-3.5,4.1,7\}/.x\_?\text{Negative}->0
\]

Similarly here, we replace all negative numbers with zero. However, notice that Negative is prefixed with a question mark. The reason for this is that Negative is a function whereas, in the earlier example, Integer is an expression type. You can use functions in your pattern matches, but they are always prefixed with a question mark. You can determine what type an expression is using the Head function, e.g. Head[5] returns Integer.

Exercises 10

1. Observe the result of the following statement:
   \[
   \{\{1,2\},\{3,4\},\{5,6\}\}/.\{p_,q_\}->\{q,p\}
   \]

2. Enter the following statement to construct and display a Graphics expression. The argument of the Line primitive is a list of pairs of real numbers specifying coordinates in two dimensions.
   \[
   g = \text{Graphics[}\{\text{Red,}\text{Line[}\{\{0,0,0\},
   \{1.0,1.0\},\{2.0,0.0\},\{3.0,2.0\},\{4.0,0.0\},\{5.0,2.0\},
   \{6.0,1.0\},\{7.0,3.0\},\{8.0,0.0\},\{9.0,2.0\},\{10.0,0.0\}\}],
   \text{Axes->True}\}]
   \]

3. Bearing in mind the switch we implemented in the exercise above, it would seem logical that we can flip the coordinate pairs in our graphic using a similar mechanism. i.e. \(g/.\{p_,q_\}->\{q,p\}\)

Why doesn’t this do what was expected?
5.14 Patterns in functions

When we first looked at defining functions, we mentioned that the underscore character was necessary, and that it defined a pattern that the argument must match in order for the function to be evaluated. For example, consider rolling two six-sided dice. The following is a function that gives the probability of achieving a particular total score from a roll of the two dice. The total can only be integer, so we define a function where the argument must be an integer:

\[
\text{prob}[n_{\text{Integer}}] := (6 - \text{Abs}[n-7])/36
\]

Exercises 11

4. Try finding the probability of rolling a total of 10. What about the probability of rolling snake-eyes?
5. Check out the return if you ask for a non-integer score.
6. Build a list of the probability of each possible scores from the two dice. Take the Total of the list and verify that it adds up to unity.
7. Of course you could try to calculate the probability of getting a non-physical score, i.e. not in the range 2 to 12. You would get a result, but it is meaningless. We can make our function definition more restrictive by adding conditions to our pattern e.g. \text{prob}[n_{\text{Integer}}/.;2 \leq n \leq 12]. In truth we rarely use these restrictions, but if you want to try this out, you should make sure to use \text{ClearAll}[\text{prob}] to remove the previous definition of the function prior to redefining it with this added condition.

Now redefine \text{f} as follows:

\[
\text{ClearAll}[f]; \text{Clear}[a]; (*Always good practice to clear key symbols*)
\]

\[
f[a_{\_}] := \text{NIntegrate}[a \ x^2, \{x, -1, 1\}]
\]

and evaluate \text{f}[2], then \text{f}[a]. When you evaluate \text{f}[a] you will get an error message as the integrand in the \text{NIntegrate} function has a non-numerical value. You have asked Mathematica to try to evaluate a numerical integral with something that does not have a numerical value. If however, we clear the definition for \text{f}, and set it up to only be defined for numerical arguments, then we get a more satisfactory, less error-prone, behaviour.

\[
\text{ClearAll}[f]; \text{Clear}[a];
\]

\[
f[a_{\_}\text{?}\text{NumericQ}] := \text{NIntegrate}[a \ x^2, \{x, -1, 1\}]
\]

\[
f[2]
\]

\[
f[a]
\]

Here we get the numerical answer we want from \text{f}[2], but when we ask for \text{f}[a] then Mathematica has no function to apply, so just echoes the input.

5.15 Solving differential equations with triggering events

When we model a system governed by a set of differential equations, we occasionally want to take note when particular events happen, or conditions are met. Looking back at the projectile problem from earlier, we calculated the numerical integral over the range \{t,0,10\}, whereas we only plotted in the range \{t,0,1\}. We limited ourselves to this plot range as beyond \(t = 1\) the projectile has a negative height, i.e. it had passed underground. Our model took no cognisance of there being a ground-level. Indeed, if you look closely, you would see that at \(t = 1\) the projectile hadn’t quite reached the ground, so this was only a very rough approximation to the time of flight.

The \text{WhenEvent} function gives us the ability to specify an action to take when a particular event, e.g. hitting the ground, \(y[t] == 0\), occurs. Let’s use \text{WhenEvent} to capture the time of flight, and plot the full motion until the projectile touches down:
Our numerical integration has limits defined as \( \{t, 0, 10\} \). If you remove the semicolon from the NDSolve expression, then you will see that the result of the numerical integration is two interpolating functions with domain \( [0, 10] \). However, after our projectile hits the ground our model is producing non-physical results. It would have been good to stop the numerical integration at touch-down. Change the WhenEvent part of the code to the following to terminate the integration once the projectile hits the ground:

\[
\text{WhenEvent}\{y[t]==0, \text{timeofflight}=t;\}
\]

Confirm that the domain on the interpolating functions is now limited to the time of flight.

### Exercises 12

8. Using the \( \text{tof}[\theta] \) function as a template, create a new function called \( \text{dof} \), that when called with \( \theta \) as an argument will return the \( x \)-range of the projectile.

9. Build a list, called \( \text{toflist} \), containing \( \theta \), time of flight and \( x \)-range for each value of \( \theta \) from 0.02 to \( \pi/2 \) radians in steps of 0.02 radians.

10. Verify that your list has dimensions \( \{78, 3\} \) i.e. 78 \( \theta \) values, each recording the list of three values: \( \{\theta, \text{time of flight}, \text{x-range}\} \).

11. Evaluate \( \text{Max}[\text{toflist}[\{\text{All}, 3\}]] \) to find the maximum \( x \)-range in the list.

12. Use the Position function to find the position of the maximum \( x \)-range in \( \text{toflist} \).

13. Find the value of \( \theta \) relating to that position in \( \text{toflist} \). How does this compare to \( \pi/4 \)? How might we get a more accurate value for the maximum range angle?

14. The steps above are a rudimentary way to find the maximum \( x \)-range. However, we can also use the FindMaximum function for this purpose. However, if you simply try to FindMaximum[\( \text{dof}[\theta], \theta \)] then you will discover that FindMaximum likes to try to initially evaluate symbolically. This means that \( \theta \) isn’t given a numerical value and the NDSolve, on which \( \text{tof} \) relies, cannot perform its numerical integration. We need to tell \( \text{tof} \) to only try to evaluate if a numerical value is passed as the argument i.e. \( \text{dof}[\theta_?\text{NumericQ}] \).
15. Clear \texttt{tof}, and redefine it to use only a numerical argument. Then use \texttt{FindMaximum[dof[\theta],\theta]} to determine the maximum $x$-range, and the associated $\theta$.

\textbf{Check the dof[x] stuff - also accuracy goal etc!}

5.17 Debugging methods

Whenever we write code we will invariably, accidentally, include bugs in the code. Whether these result from a mistyping of upper/lower case, or maybe a missing comma, they can be frustrating to try to find. Many of the most common errors can be found by observing Mathematica’s colouring arrangements. Others errors are more subtle. Use \texttt{NotebookOpen[path <> "intro/abird.nb"]} to open a notebook containing a block of code solving the trajectory of a projectile. The equation of motion is given by:

\[
m\frac{d^2 \vec{r}}{dt^2} = m\vec{g} - \gamma \left( \frac{d\vec{r}}{dt} - \vec{w} \right)
\]

where $m = 0.2\text{kg}$, $\vec{g} = (0, -9.81\text{ms}^{-2})$ and $\gamma = 0.03\text{kg/s}$ are the mass, the gravitational field, the wind speed and the air drag coefficient, respectively. The scope of the code is to create an interactive manipulate that shows the effect of the initial velocity (amplitude and angle) and horizontal wind speed on the trajectory of the "angry bird". In this code we have included six errors. Can you find them all? The final output should look like:

In general, the best approach to debugging is to check each line individually and sequentially in their own cell. Make sure that each function has the input it is expecting and delivers the expected output in the right format. Once this is done, merge all cells back together and re-check the whole code. You should check the code with a newly launched kernel, e.g. quit Mathematica and restart - this means you have to save also!

**Alphabetical list of further functions met**

\[
\text{explicitlyUsed} = \{\text{ClearAll, Condition, FindMaximum, Function, Grid, ImageData, Line, Map, Max, Module, Monitor, NMinimize, NotebookOpen, Pause, Position, Graphics Raster, StringLength, StringPosition, StringReplace, StringTake, Sum, ToString, Total, WhenEvent}\};
\]
6 Celestial mechanics (week 3)

6.1 Programming techniques

Solving differential vector equations Initially, a basketball is thrown from the top of a 50 m building horizontally at 30 m s\(^{-1}\). The following numerical integration solves the motion for the position vector \(\vec{r}(t) = (x(t), y(t))\) in the first ten seconds.

\[
\begin{align*}
m &= 0.6; (* \text{ kg } *) \\
\text{s00} &= \text{NDSolve}\{m \vec{r}''[t] == m \{0, -9.81\}, \vec{r}[0] == \{0, 50\}, \vec{r}'[0] == \{30, 0\}, \vec{r}, \{t, 0, 10\}\}[1]; \\
\text{ParametricPlot}[\vec{r}[t] /. \text{s00}, \{t, 0, 10\}, \\
\text{AxesLabel} -> \{"x (m)", "y (m)"\}, \text{PlotLabel} -> \"Trajectory\"]
\end{align*}
\]

1. Define two vectors: \(a = \{1, 2\}\) and \(b = \{-3, 4\}\). Calculate and compare \(a \cdot b\) and \(a \times b\).

2. Debug and correct the \texttt{Plot} function below to show the kinetic energy of the ball as a function of time.

3. Find the position of the thrown basketball at \(t = 8\) s.

\[
\begin{align*}
\text{Plot}\left[\frac{1}{2} \text{m} \vec{r}[t].\vec{r}[t] /. \text{s00}, \{t, 0, 10\}, \\
\text{AxesLabel} -> \{\"t (s)\", \"Energy (J)\"\}, \text{PlotLabel} -> \"Kinetic energy\"]
\end{align*}
\]

Conditions inside differential equations Simulate a bouncing ball that retains 70% of its velocity in each bounce:

\[
\begin{align*}
g &= \{0, -9.81\}; m = 0.4; \\
\text{s01} &= \text{NDSolve}\{m \vec{r}''[t] == m g, \vec{r}[0] == \{0, 50\}, \vec{r}'[0] == \{30, 0\}, \\
\&\& \text{WhenEvent}[\vec{r}[t][[2]] == 0, \vec{r}'[t] -> \{1, -.7\} \vec{r}'[t]], \vec{r}, \{t, 0, 10\}\}[1]; \\
\text{ParametricPlot}[\vec{r}[t] /. \text{s01}, \{t, 0, 10\}, \\
\text{AxesLabel} -> \{"x (m)", "y (m)"\}, \text{PlotLabel} -> \"Trajectory\"]
\end{align*}
\]

4. Use the function \texttt{Norm} to plot \(|\vec{r}'(t)|\) as a function of time. What does it mean?

5. Use the code below to find the time of the first bounce (called \texttt{hit}). What does the [2] mean, and why is it desirable to use it here? Plot the trajectory to this time.

\[
\begin{align*}
\text{s02} &= \text{NDSolve}\{m \vec{r}''[t] == m g, \vec{r}[0] == \{0, 50\}, \vec{r}'[0] == \{30, 0\}, \\
\&\& \text{WhenEvent}[\vec{r}[t][[2]] == 0, \text{hit} = t; \text{"StopIntegration"}], \vec{r}, \{t, 0, 10\}\}[1];
\end{align*}
\]

Parameter variations in differential equations When you execute the code below you will get an error saying a number “cannot be used as a variable”. This is due to \texttt{Plot} setting a numerical value for \(t\) which \texttt{NDSolve} then tries to use as a variable.

\[
\begin{align*}
pfun[a_, t_] := \text{Module}[\{\text{sol}\}, \\
\text{sol} = \text{NDSolve}[\{f''[t] + a f[t] == 0, f[0] == 1, f'[0] == 0\}, f, \{t, 0, 10\}\}[1]; \\
f[t] /. \text{sol}
\end{align*}
\]

\[
\begin{align*}
\text{Plot}[pfun[2, t], \{t, 0, 10\}, \\
\text{AxesLabel} -> \{\"t (arb.u.)\", \"position (arb.u.)\"\}, \text{PlotLabel} -> \"f(t)\"]
\end{align*}
\]

6. If you ask Mathematica to evaluate \(pfun[2, t]\) prior to numerical values of \(t\) being set by the \texttt{Plot} function, then the “cannot be used as a variable” error can be avoided. Do this by changing the first argument in \texttt{Plot} to \texttt{Evaluate[pfun[2, t]]}.

7. Alternatively, you can change the variable name \(t\) in \texttt{NDSolve} to the dummy variable \(\tau\) and plot without having to use \texttt{Evaluate}.

8. Make the plot interactive using the \texttt{Manipulate} function such that it shows the variation of the parameters \(a\) in the range 0.5 to 3.
Special plots  The following code plots the function \( f(x) = x^3 \) using the four different logarithmic representations.

\[
f[x_] := x^3
\]

```
GraphicsGrid[{{Plot[f[x], {x, 1, 6}, PlotLabel -> "Linear-Linear"],
               LogLinearPlot[f[x], {x, 1, 6}, PlotLabel -> "Log-Linear"],
               LogPlot[f[x], {x, 1, 6}, PlotLabel -> "Linear-Log"],
               LogLogPlot[f[x], {x, 1, 6}, PlotLabel -> "Log-Log"]}}]
```

9. Add to each of the graphs a point with coordinates (5,125).

Fitting data  This code makes a list of the y-positions of the basketball until it hits the ground. We have added some random variation to the y-values, as measured data generally has random uncertainties attached. We fit and plot data with the model function \( y(t) = a_0 + a_1 t + a_2 t^2 \).

```
ypos = Table[{t, RandomReal[] + (r[t] /. sol2)[[2]]}, {t, 0, hit, hit/100}];
model = a0 + a1 t + a2 t^2;
fit = FindFit[ypos, model, {a0, a1, a2}, t];
Plot[model /. fit, {t, 0, hit}, Epilog -> Point[ypos]]
```

10. Add the best fit function as a plot label, and label the axes.

6.2 Review of physics background

Celestial mechanics mainly deals with the description of the motion of planets and other celestial bodies. The goal of this study is to describe the motion of the planets around the Sun and to check Kepler’s third Law.

Newton’s force of gravity acting on the Earth is defined by

\[
F_e(r(t)) = m_e \frac{d^2r(t)}{dt^2} = -G \frac{m_e m_s}{r(t)^2} \frac{\dot{r}(t)}{|\dot{r}(t)|} \tag{6.1}
\]

where \( G = 6.6725 \cdot 10^{-11} \) kg\(^{-1}\) m\(^3\) s\(^{-2}\) is the gravitational constant, \( r \) is the position vector of the Earth with respect to the Sun, \( m_s = 1.989 \cdot 10^{30} \) kg the mass of the Sun and \( m_e = 5.972 \cdot 10^{24} \) kg the mass of the Earth.

6.3 Study

11. Solving the equation of motion

In a first step we will solve the equation dealing with the motion of the Earth around the Sun. Given that the mass of the Sun is so large in comparison to the mass of the Earth, we will make a simplifying assumption that the Sun simply stays fixed in one spot, the origin \( r_s=(0,0) \). Using NDSolve and the equation (6.1) above, we can compute the orbit of the Earth for 3 years. To do this we also need to specify an initial position and an initial velocity for the Earth. The following list contains the mean distance from the sun to the different celestial objects in astronomical units (1 AU=149 597 870 700 m). The third column contains their respective initial velocity in m s\(^{-1}\).

12. Visualising the orbits

The function ParametricPlot can be used to plot the trajectory of the Earth for the first 3 years. Visualise the orbit of the Earth using this command.

<table>
<thead>
<tr>
<th>Object</th>
<th>Orbital radius (AU)</th>
<th>Initial tangential velocity (m s(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mercury</td>
<td>0.38697</td>
<td>47880.1</td>
</tr>
<tr>
<td>Venus</td>
<td>0.722603</td>
<td>35038.8</td>
</tr>
<tr>
<td>Earth</td>
<td>1</td>
<td>29785.1</td>
</tr>
<tr>
<td>Eros*</td>
<td>1.449887</td>
<td>24736.1</td>
</tr>
<tr>
<td>Mars</td>
<td>1.522748</td>
<td>24137.1</td>
</tr>
<tr>
<td>Ceres*</td>
<td>2.76675</td>
<td>17906.6</td>
</tr>
<tr>
<td>Jupiter</td>
<td>5.19994</td>
<td>13061.7</td>
</tr>
<tr>
<td>Saturn</td>
<td>9.565644</td>
<td>9630.4</td>
</tr>
<tr>
<td>Uranus</td>
<td>19.27835</td>
<td>6783.7</td>
</tr>
<tr>
<td>Neptune</td>
<td>30.13411</td>
<td>5425.9</td>
</tr>
<tr>
<td>Pluto</td>
<td>39.87356</td>
<td>4716.9</td>
</tr>
</tbody>
</table>

Table 1: Planet parameters - adjusted for eccentricity and inclination (*asteroid).
13. **Escape Velocity visualisation**

Define a function to solve the motion of the Earth with the initial speed as a parameter. You might want to revisit the “Differential equation solving” (section 5.4) and the “Solving differential equations for parameteric study” (5.10) sections in the introduction to get started. When solving use the option `AccuracyGoal→1` to set the accuracy of the integration to one decimal place. Create a table of 5 solutions with the initial speed of the Earth varying from 50% to 150% in steps of 25%. Plot all five trajectories on a single graph for a duration of 3 years.

14. **Planet orbital period function**

Using `WhenEvent` and `StopIntegration`, determine the period of the orbit of the Earth. Define a function that gives the period as a function of initial position and initial velocity.

15. **Kepler’s Third Law**

Using the orbital period function defined earlier and the planet data provided, calculate the period of the different celestial bodies in the list. You might find it useful to make a list, maybe called `planetList` that hold the data from Table 1. Note that the mass of each planet cancels out in the equations of motion of each individual planet. Create a Log-Log plot of the orbital period versus the orbital radius of the celestial bodies and check on the same graph against Kepler’s Third Law stating: \( T = \sqrt{\frac{r^3}{Gm_s}} \). Fit the period versus radius data using Kepler’s Law to determine the power, \( b \) and multiplicative factor \( a \), e.g. \( a r^b \).


### 6.4 Advanced problems

You might consider further studies looking into 2-body and 3-body interactions. The motion of whole galaxy colliding with each other can be simulated using approaches such as Barnes-Hut algorithm.
7 Galaxy redshift (week 4)

7.1 Programming techniques

**Doppler shift** Here, we consider a sound wave travelling in the negative \( x \)-direction \( c = 343.5 \text{ m/s} \). The sound wave comprises a Gaussian amplitude envelope, characterised by a width parameter \( dx = 50.0 \text{ m} \), and a carrier wave with a frequency of 300 Hz. The observer is moving with the velocity \( v_0 = 100.0 \text{ m/s} \) and samples the wave with a sample rate \( s = 5000.0 \text{ Hz} \) for a duration of 1.0 s. You might think of this as a microphone, attached to an oscilloscope, travelling at \( v_0 \) being sampled by the oscilloscope \( s \) times per second.

\[
\begin{align*}
  c &= 343.5; \\
  dx &= 50.0; \\
  f_0 &= 300.0; \\
  v_0 &= 100.0; \\
  \text{source}[x_-, t_] &= \exp[-(x + c t)^2/dx^2] \cos[2 \pi f_0 (t + x/c)] \\
  \text{xp}[t_] &= v_0 t; \\
  s &= 5000.0; \\
  \text{ts} &= \text{Table[source[xp[t], t], \{t, -T/2, T/2, dt\}];}
\end{align*}
\]

1. First, use `ListLinePlot` to plot the entirety of the detected signal, then limit the `PlotRange` to show approximately 10 periods in the centre of the pulse.
2. Use `FindPeaks` to detect the positions of the maxima.
3. Use `Differences` to create a list of the period between each successive maxima and calculate the average frequency observed. Compare with the expected Doppler shifted frequency given by \( \Delta f = \frac{v_0 c}{c} f_0 \).
4. Analyse the code below to see how it works, and compare with the previous period-finding approach.

\[
\begin{align*}
  \text{pos} &= \text{Table[If[ts[[i]] ts[[i - 1]] < 0 \&\& ts[[i]] > 0, i, Missing[]], \{i, 2, \text{Length[ts]}\}];} \\
  \text{period} &= \text{dt Mean[Differences[DeleteMissing[pos]]];}
\end{align*}
\]

**Importing ASCII data and fitting** Experiments often generate digital data in different formats. The ASCII data format is one of the most generic file format used. Here, we import a remote data file, extract its header defining the data and fit the data to a model.

\[
\begin{align*}
  \text{path} &= "https://www.st-andrews.ac.uk/physics/ph3080/"; \\
  \text{rawdata} &= \text{Import[path <> "galaxies/spectrum.dat";} \\
  \text{rawdata[[1 ;; 2]]} &= \mathbf{\text{TableForm}} \\
  \text{dat} &= \text{rawdata[[3 ;; -1]]};
\end{align*}
\]

5. Use `ListLinePlot` to plot the imported data.
6. Use `FindPeaks` to measure peak position in Hz. Note that `FindPeaks` has a number of optional parameters that sometimes need to be set in order for the correct peaks to be found. Experiment with the different settings. As the peaks are well defined we can set the Gaussian blurring and sharpness to zero and use the threshold value to pick the correct peaks.
7. Use `FindFit` to fit the data to \( a \exp(-(f - f_d)^2/w^2) \) with \( a, f_d \) and \( w \) as the fitting parameters and the frequency, \( f \), as the variable.
8. Plot the fit and imported data together with appropriate labels and scale. Use `GridLines` to indicate the position of the peak.

**Importing FITS files** A very common file format used in astronomy is the Flexible Image Transport System (FITS). Here, we use this format to import a 3D datacube from the imaging of galaxy NGC 7591.

\[
\begin{align*}
  \text{\{meta, data\} &= \text{Import[path <> "galaxies/NGC7591_gas_nonoise.fits",} \\
  \text{\{"Metadata", "RawData"\}\} &= \mathbf{\text{Normal}};}
\end{align*}
\]
9. Determine the dimensions of the data and the wavelength list.
10. Using a replacement rule, set variables units and name to the units and object name used in the FITS header.
11. Use function Total to sum up all the spectral components and create a grey scale image of the galaxy. The Image and Rescale functions are useful.

### 7.2 Physics background

Integral Field Spectroscopy (IFS) is the latest technique that is helping us understand how galaxies form and evolve. A grid of optical fibres are placed within the field-of-view of the telescope, with each one leading to a spectrograph where the light is dispersed. The result is a 3D image of the galaxy: two spatial dimensions and one spectral (wavelength) dimension. From these “datacubes” we can measure the spatially resolved dynamics of the gas and stars in galaxies, we can see where most of the star formation is occurring, and we can look for spatial variations in star formation history and metal content.

In this case study you will be given a noise-filtered datacube of the galaxy NGC 7591 which has been observed as part of the ongoing CALIFA survey, and use it to measure some of the properties of the galaxy. For our study, we need to consider the cosmological redshift $z$ due to the expansion of the universe. This redshift describes the wavelength change of a wave when the space between source and observer is expanding. The expression involves the measured wavelength $\lambda_m$ and the emission wavelength $\lambda_0$.

$$ z = \frac{\lambda_m}{\lambda_0} - 1 $$

Also useful to our study is Hubble’s Law which describes the expansion of the Universe and links the distance to deep space objects $D$ and their recession velocity $v_r$ via Hubble’s constant $H_0 = 67.7$ (km/s)/Mpc, where Mpc are distance measurements units corresponding to megaparsecs (1 Mpc=3.09×10$^{19}$ km).

### 7.3 Study

In this study, we want to determine the recession velocity, and distance to, the NGC 7591 galaxy.

12. Determine the total spectrum of the whole galaxy image. Plot this, scaling the x-axis with the wavelengths from obswave. You might find page 20 useful.
13. Identify the main lines in the spectrum using the table provided. You should look at two wavelength regions (480nm, 520nm) and (650nm, 680nm). Note that 10 Å=1 nm.

<table>
<thead>
<tr>
<th>Name</th>
<th>$H_\beta$</th>
<th>O\textsc{iii}</th>
<th>O\textsc{iii}</th>
<th>N\textsc{ii}</th>
<th>H$\alpha$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wavelength (nm)</td>
<td>486.3</td>
<td>496.0</td>
<td>500.8</td>
<td>655.0, 658.5</td>
<td>656.4</td>
</tr>
</tbody>
</table>

14. Estimate the cosmological redshift by detecting the measured peaks using FindPeaks.
15. Use FindFit to fit three Gaussian peaks to the H$\alpha$ and two N\textsc{ii} lines and determine the redshift of the galaxy. Use the estimated redshift as a starting position for the fit.
16. The cosmological redshift is linked to object recession velocity via: $v_r = z \cdot c$ where $c$ is the speed of light. Determine the recession velocity of the galaxy and, assuming Hubble’s law, estimate the distance to the galaxy. You can compare the fitted redshift to that from the CALIFA survey $z_c = 0.0165$. 

41
17. Using the same fitting procedure, determine the total redshift for the points of the galaxy at image coordinates (33,33) and (42,42). The shift in lines at these points is caused by the sum of the cosmological redshift of the galaxy and the Doppler shift caused by the motion of the gas in the galaxy.

18. Subtract the cosmological redshift component from the total redshift to determine the Doppler shift. Using this Doppler shift, determine the local velocity of these two points with respect to the galaxy frame of reference.

7.4 Advanced problem

Study the distribution of the local velocity as a function of the distance from the centre of the galaxy. Does this look like what you would expect and how does this relate to Dark Matter?
8 Double pendulum (week 5)

8.1 Programming techniques

If you copy and paste from the pdf of the course notes, then you must closely check the pasted code. Certain symbols, spacing and return characters do not always copy across as expected.

Single pendulum The single pendulum can be solved numerically using the following code.

\[
m_1 = 1.5; \quad L_1 = 1.2; \quad g = 9.81;
\]

\[
eq 1 = \{\theta_1''(t) = -g/L_1 \sin(\theta_1(t))\};
\]

\[
\text{init} = \{\theta_1[0] = 2, \ \theta_1'[0] = 0\};
\]

\[
sol1 = \text{NDSolve}[\{\text{eq1, init}, \theta_1, \{t, 0, 10\}\}[1]
\]

1. Plot the angle \(\theta_1(t)\) and the angular velocity \(\theta_1'(t)\) as a function of time on separate graphs. Label the graphs appropriately.

2. Use a plot to verify that the total energy is conserved.

Animation We define a function for the position of the bob at a given time, and a second function that shows the position of the bob, and plots its trajectory up to a given time.

\[
r1[t_] := L_1 \{\sin(\theta_1[t]), -\cos(\theta_1[t])\} /. \text{sol1}
\]

\[
\text{path}[tm_] := \text{ParametricPlot}[r1[t], \{t, 0, tm\},
\text{Epilog} \rightarrow \{\text{Blue, Disk}[r1[tm], 0.1], \text{Line}[\{\{0, 0\}, r1[tm]\}]\},
\text{PlotRange} \rightarrow 1.1 \{\{-L_1, L_1\}, \{-L_1, L_1\}\}, \text{Frame} \rightarrow \text{True}, \text{PlotPoints} \rightarrow 60]
\]

3. Animate the path taken by the pendulum bob.

Phase space To study our system we will use a phase-space representation (angular velocity \(\theta_1'(t)\) versus angular position \(\theta_1(t)\)).

\[
sol2 = \text{NDSolve}[\{\text{eq1, init}, \theta_1, \{t, 0, 10\}\}[1];
\]

\[
\text{ParametricPlot}[\{\text{Mod}[\theta_1[t], 2 \pi, -\pi], \theta_1'[t]\} /. \text{sol2},
\{t, 0, 10\}, \text{Frame} \rightarrow \text{True}, \text{FrameLabel} \rightarrow \{"\theta_1", "\theta_1'", "Phase space"}\]
\]

4. To better understand the use of \text{Mod} in the above code, use a plot to check the effects of \text{Mod}[x + \pi, 2\pi] - \pi on variable \(x\) in the range \(-5\pi\) to \(5\pi\).

5. Create an interactive manipulation showing the phase space plot while changing the initial angular velocity of the bob between \(-\pi\) to \(\pi\).

Poincaré maps This can be used to show the onset of chaotic motion. These maps correspond to phase space plots sampled at regular or specific intervals.

\[
tmax = 1000;
\]

\[
sol3 = \text{NDSolve}[\{\text{eq1, init}, \theta_1, \{t, 0, tmax\}\}[1];
\]

\[
\text{pt} = \text{Table}[\{\text{Mod}[\theta_1[t], 2 \pi, -\pi], \theta_1'[t]\} /. \text{sol3},
\{t, 0, \text{tmax}, 2 \pi \text{Sqrt}[L_1/g]\}];
\]

\[
\text{ListPlot}[\text{pt, Frame} \rightarrow \text{True, FrameLabel} \rightarrow \{"\theta_1", "\theta_1'", "Poincare section"\}]
\]

6. Create a superposition of Poincaré maps, each corresponding to an initial angular velocity varying between \(-8\) rad/s and \(8\) rad/s in steps of 0.5 rad/s. Each Poincaré map should have a different colour.
8.2 Review of background physics

In this study we will model the motion of a double pendulum. The equations of motion for a double pendulum consist of a system of two coupled equations. We have:

\[ m_1 \ddot{r}_1 = -\frac{T_1}{L_1} r_1 + \frac{T_2}{L_2} (r_2 - r_1) + m_1 g \]
\[ m_2 \ddot{r}_2 = -\frac{T_2}{L_2} (r_2 - r_1) + m_2 g \]

with positions vectors \( r_1 = L_1 (\sin(\theta_1), -\cos(\theta_1)) \) and \( r_2 = r_1 + L_2 (\sin(\theta_2), -\cos(\theta_2)) \). This leads altogether to four equations and four unknowns: \( \theta_1, \theta_2, T_1 \) and \( T_2 \). Eliminating the internal tensions in the equations delivers us two coupled differential equations only involving \( \theta_1 \) and \( \theta_2 \).

\[
(m_1 + m_2) L_1 \theta_1'' + m_2 L_2 \theta_2'' \cos(\theta_1 - \theta_2) + m_2 L_2 \theta_2'^2 \sin(\theta_1 - \theta_2) + g (m_1 + m_2) \sin(\theta_1) = 0 \\
m_2 L_2 \theta_2'' + m_2 L_1 \theta_1'' \cos(\theta_1 - \theta_2) - m_2 L_2 \theta_2'^2 \sin(\theta_1 - \theta_2) + m_2 g \sin(\theta_2) = 0
\]

8.3 Study

Here, we want to study the simplest case of the double pendulum where the two arm lengths \( L_1 = L_2 = 1.2 \) m and the two masses \( m_1 = m_2 = 1.3 \) kg are equal to each other (\( g = 9.81 \) m/s\(^2\)).

7. Numerically solve the equations of motion of the double pendulum for the first 10 seconds, using the initial conditions \( \theta_1(0) = 1, \theta_2(0) = 0 \) and \( \theta_1'(0) = \theta_2'(0) = 0 \).
8. Animate the path taken by the two pendulum bobs.
9. Use a plot to verify that the total energy is conserved. Plot the total energy along with the kinetic and potential energy. Define the potential energy with respect to the position \( r = (0, -L_1 - L_2) \).
10. Create a phase space plot and Poincaré map of the first bob at equal intervals \( T = 2\pi \sqrt{L_1/g} \) for the first 1000 second.
11. Modify the Poincaré map to record whenever the second bob’s angular position changes sign from negative to positive. Use `WhenEvent` \[ \theta_2[t] == 0 \&\& \theta_2'[t] > 0 \] \( \ldots \) with `AppendTo` to build a list of the map points \( \{\theta_1, \theta_2\} \). This `WhenEvent` is triggered by the event \( \theta_2[t] == 0 \) and filtered using the condition \( \theta_2'[t] > 0 \).
12. Write an expression for the total energy in terms of \( \theta_1, \theta_2, \omega_1 \) and \( \omega_2 \). You should be sure that the variables do not have numerical values so that you end up with a symbolic expression. Use `Solve` to express \( \omega_2 \) in terms of the other parameters.
13. Create a superposition of Poincaré maps, each corresponding to initial angular velocity of the first bob varying between -4 rad/s and 4 rad/s in steps of 0.5 rad/s. The initial angular position of the first and second bob is zero and the initial angular velocity of the second bob is chosen such that the total energy is 30 J (use the expression for \( \omega_2 \) from the previous point). In order for the Poincaré maps to be fairly compared, each map must be from a double-pendulum system with the same total energy. Each Poincaré map should have a different colour.
14. Investigate the case of the total energy equal to 35 J.

8.4 Advanced problem

Expand the model to include a third pendulum and study its trajectory.
9 Independent learning week (week 6)

Use this week to consolidate your learning by reviewing the material from the introduction and the first three case studies including associated Homework. You can also use this time as an opportunity to tackle the “Advanced problem” sections in the case studies. All this will help you become fluent with Mathematica and its application to physics and astronomy.

9.1 Translating programming languages

Many of the techniques we use in Mathematica can be transferred to other programming languages. This site [http://rosettacode.org/wiki/Rosetta_Code](http://rosettacode.org/wiki/Rosetta_Code) offers a comparison between the different programming languages. You can use this site to understand how to translate between different programming languages. The Mathematica webpage can be found here [http://rosettacode.org/wiki/Category:Mathematica](http://rosettacode.org/wiki/Category:Mathematica).

9.2 Small optional project

"2006RH120 (formerly: 6R10DB9) is a tiny near-Earth asteroid and fast rotator with a diameter of approximately 2-3 meters that ordinarily orbits the Sun but makes close approaches to the Earth-Moon system around every twenty years, when it can temporarily enter Earth orbit through temporary satellite capture (TSC). Most recently, it was in Earth orbit from September 2006 to June 2007." [http://www.knowpia.com/pages/2006_RH120](http://www.knowpia.com/pages/2006_RH120)

Using the techniques developed in the celestial mechanics case study you can simulate the trajectory of Near Earth Object 2006 RH120 from September 2006 to June 2007. The initial positions and velocities for the Earth, Moon and 2006RH120 can be found at [https://ssd.jpl.nasa.gov/horizons.cgi](https://ssd.jpl.nasa.gov/horizons.cgi). This screenshot shows the pertinent choices to setup useful output. Make sure you are aware that the units are length in AU and velocity in AU/day.
10 Fourier (week 7)

10.1 Review of background

There is a mathematical theorem that states that any continuous time series can be represented as a sum of sinusoids of different frequencies with different amplitudes (and phases), and that this is a completely equivalent representation. In order to determine this representation we use the Fourier transform. The Fourier transform involves the term \( \exp(i\omega t) \) which can be thought of as \( \cos(\omega t) + i\sin(\omega t) = \exp(i\omega t) \).

In experimental physics, we often record the output of a sensor over a period of time at a fixed sampling rate. In this case we use the discrete version of the Fourier transform which assumes the signal to be periodically repeating beyond the measurement duration.

Once you have learned about Fourier transforms, you might see how these could be used in our earlier case studies, e.g. to calculate the shifted frequency in the Galaxy redshift, or to determine resonant frequencies in the Double pendulum.

10.2 Programming techniques

Create a time series Suppose we were measuring a voltage that consisted of a sinusoidal wave of frequency \( f = 5 \) Hz, with amplitude 1 V, measured every 5 ms for 161 readings.

```math
numpt = 161; dt = 0.005; f = 5;
data = Table[Cos[2 Pi f t + Pi/3], {t, 0, dt (numpt - 1), dt}];
Length[data]
ListPlot[data, PlotRange -> All, DataRange -> {0, dt (numpt - 1)},
PlotStyle -> Red, Frame -> True,
FrameLabel -> {"Time (s)", "Voltage (V)"}, PlotLabel -> "Data"]
```

1. Plot the discrete data points alongside the continuous function \( \cos(2\pi ft + \pi/3) \).
2. Use the Join function to join together three copies of data. Verify that the new list length (numpt) is three times the original. Plot this enlarged list alongside the continuous function. Explain why this list of data points is not periodic when repeated.
3. Modify data to make it periodic and call the new list data2.

Take the Discrete Fourier Transform Use the Fourier function to determine the discrete Fourier transform of the signal. We then plot the absolute value of the calculated discrete Fourier transform.

```math
fdata = Fourier[data];
```

4. The Fourier data is usually complex. Use ListLinePlot to visualise the Fourier components as absolute values (Abs[fdata]). Note the two unambiguous frequency components.
5. Redo the above with the periodic data (data2).
6. Look at the values in fdata for the periodic data. Notice that you have generated a list of complex numbers, albeit most of them are zero (or near zero) for this particular case. Use the Chop function to tidy up the Fourier data so you can better see the non-zero components. This list of complex numbers correspond to the amplitudes and phases of the Fourier components. Note that Chop is use here only for visualisation purposes. This function in not part of the Fourier transform process.
7. Use the function TrigToExp to convert the expressions \( \sin(\omega t) \) and \( \cos(\omega t) \) into their exponential equivalents. Note the number of components of each and the frequencies associated with these components.
8. In the same way that you can extract elements from a list using the Part function, you can also extract parts of expressions. Verify this by using the Part function to extract each of the parts of TrigToExp expanded \( \sin \) & \( \cos \) expressions into two lists: \( \{1/2 e^{-i\omega t}, 1/2 e^{i\omega t}\} \) & \( \{1/2 i e^{-i\omega t}, -1/2 i e^{i\omega t}\} \).
9. Since the data is assumed to be real and periodic, the second peak corresponds to a negative frequency. Check this statement visually by plotting the amplitudes of the Fourier transform of \(\exp(-2\pi if t)\) for frequency \(f = \pm 5\) using the same sampling as in point 1.

**Correct frequency scale**  
The correct scale in the frequency domain is inversely proportional to the overall time-duration of the acquisition, \(T\). This means that the frequency difference between two elements in the Fourier list (components) is defined by \(\Delta f = 1/T\). Note that the first element of the Fourier transform has a frequency of zero hertz and that the second half of the list represents negative frequencies with the last element having a frequency of \(-\Delta f\).

\[
T = dt \cdot \text{Length}[\text{data2}];
\]
\[
df = 1/T; \quad (* \text{frequency step} *)
\]
\[
nzoom = \text{Round}[\text{fmax}/\text{df}]; \quad (* \text{number of points in the Fourier list required} *)
\]

```math
\begin{align*}
\text{GraphicsGrid} & \{\{\text{ListLinePlot}[\text{data2}, \text{PlotRange} \to \text{All}, \text{Frame} \to \text{True}, \text{FrameLabel} \to \{"Time (s)\", "Voltage (V)\"}, \\
 & \quad \text{PlotLabel} \to \text{"Signal"}], \{\text{ListLinePlot}[\text{Abs}[\text{fdata2}[[1 ;; nzoom]]], \text{DataRange} \to \{0, (nzoom - 1)\cdot df\}, \text{PlotRange} \to \text{All}, \text{Frame} \to \text{True}, \\
 & \quad \text{FrameLabel} \to \{"Frequency (Hz)\", "Abs[Voltage] (V)\"}, \text{PlotLabel} \to \text{"Discrete Fourier transform"}]\}}
\end{align*}
```

10. Use `FindPeaks` to determine the frequency, in hertz, of the Fourier peak.
11. Use `InverseFourier`, to transform the list of Fourier components back to the time domain. Check the output against the original data `data2`, and find the maximum difference (error).

**Superposition of oscillations and noise filtering**  
Create a new time series consisting of the sum of two sinusoidal waves with frequencies \(f_a = 20\) Hz and \(f_b = 30\) Hz. Add noise to this time series and plot the noisy data.

\[
\begin{align*}
\text{numpt} &= 160; \quad \text{dt} = 0.005; \quad T = \text{dt} \cdot \text{numpt}; \quad f_{a3} = 20; \quad f_{b3} = 30; \quad \text{noiselevel} = 1; \\
\text{noise} &= \text{noiselevel} \cdot \text{RandomReal}[\{-1, 1\}, \text{numpt}]; \\
\text{cleandata} &= \text{Table}[	ext{Cos}[2 \cdot \text{Pi} \cdot f_{a3} \cdot t] + \text{Cos}[2 \cdot \text{Pi} \cdot f_{b3} \cdot t], \{t, 0, \text{dt} \cdot (\text{numpt} - 1), \text{dt}\}]; \\
\text{data} &= \text{cleandata} + \text{noise}; \\
\text{ListLinePlot} &\quad \text{data, \text{PlotRange} \to \text{All, \text{DataRange} \to \{0, \text{dt} \cdot (\text{numpt} - 1)\}, \quad \text{PlotStyle} \to \text{Red, \text{Frame} \to \text{True,} \\
 & \quad \text{FrameLabel} \to \{"Time (s)\", "Voltage (V)\"}, \text{PlotLabel} \to \text{"Noisy data"}]}
\end{align*}
```

12. Fourier transform and plot the spectrum of the noisy data.
13. Threshold filters are used when the amplitude of the signal is well above the noise level, as is the case here. Use `Threshold` to filter out all but the distinct Fourier peaks, inverse Fourier and compare to the `cleandata`, finding the maximum difference (error).
14. Create a Fourier band-pass filter that eliminates the noise in the spectral domain and inverse Fourier transform the filtered spectrum to create a filtered time series. In this case, the filter consists of a list of zeros and ones. The ones should be positioned so as to retain the peaks that we want when the filter is multiplied with the spectral data. The unwanted information is zeroed out through filtering. Compare graphically the original and the filtered time series. Band pass filters are used when the frequency domain of the signal is known.

10.3  **Study**

Amplitude modulation (AM) is commonly used in the transmission of data, e.g. AM radio. In this study, we want to give you a sense of how AM works, and what it looks like in both the time (temporal) and frequency domains.
A real signal, for example, a voice transmission from a telephone call, might consist of a signal covering a band of frequencies from 100 Hz to 4 kHz whose amplitude and phase would vary with time. A high quality radio music signal might use a wider bandwidth, say from 20Hz to 20kHz. Now, for multiple telephone signals to be transmitted down the same transmission line, or to have multiple radio stations transmitting simultaneously, all these signals are up-converted (frequency translates up) to different higher frequencies. This is done using amplitude or frequency modulation of a higher frequency, called a carrier frequency. Your radio then selects this carrier frequency, depending on your choice of radio station, and down-converts (frequency translates down) to create an electronic signal that can then be played on speakers to create the audio output. The same principal works for TV and the internet, but is a little more complex.

**AM time series and its Fourier transform** Define a modulation function \( m(t) = a_1 \sin(\omega_1 t) + a_2 \sin(\omega_2 t) \) and a carrier oscillation \( c(t) = a_c \cos(\omega_c t) \) with \( \omega_1 = 2\pi f_1 \), \( \omega_2 = 2\pi f_2 \) and \( \omega_c = 2\pi f_c \). The AM signal is then defined by \( \text{sig}(t) = (1 + m(t))c(t) \). To keep things simple, and to focus on understanding what is going on, we will just consider \( f_1 = 20 \) Hz and \( f_2 = 25 \) Hz tones as our signal, having amplitudes of \( a_1 = 0.45 \) and \( a_2 = 0.2 \). We will make our carrier frequency \( f_c = 200 \) Hz (although in practice carrier frequencies are generally much higher) with an amplitude of \( a_c = 2 \).

15. Create three time series (regularly sampled signals) corresponding to the carrier wave, the modulation signal and amplitude modulated signal with a sampling rate of 2000 Hz for 1200 values. Plot these time series.
16. Fourier transform the amplitude-modulated signal and visualise the frequencies up to 300 Hz.
17. To work with an analytical expression of the AM signal, use \texttt{Clear} to remove the numerical values from variables \( a_c, a_1, a_2, \omega_c, \omega_1 \) and \( \omega_2 \). Use \texttt{TrigToExp} and \texttt{Expand} to convert the AM signal into a sum of exponential terms. Use \texttt{FindPeaks} to detect all peaks in the numerical Fourier domain and identify the analytical exponential terms to these peaks.

**Digital demodulation technique** Digital demodulation is a numerical procedure which, through filtering and digital frequency shifting, reconstructs the original modulation signal.

18. Create a filter that only retains the positive frequency components and that filters out the carrier frequency. This is similar to the pass-band filter in point 14, however now this filter is centered around the carrier frequency.
19. Shift the filtered Fourier components in frequency such that the carrier frequency is at zero. This can be achieved using \texttt{RotateLeft}.
20. Inverse Fourier and compare the resulting real part with the original modulation function. Where does the factor of 2 come from?

**Classical demodulation technique** An electronically simple method to demodulate AM signal is through the use of a non-linear detector measuring \( \text{sig}^2 = s(t)^2 \). We then pass the output through a low-frequency pass filter. This results in the amplitude envelope which is the original signal.

1. Using the previously defined analytical AM signal, square it and convert it to a sum of exponential terms. Use the \texttt{Length} function to verify that there are 45 terms in this sum. We are interested in the terms that do not involve any carrier frequency, so we could use the \texttt{Part} function to manually extract these from the 45 terms. However, it is more effective to use

\[
\text{Select[\text{sig}^2, \text{Length}[\text{Position}[#, \omega_c]] == 0 &]}
\]

 to select only the terms that do not involve any carrier frequency. Select the terms and simplify the resulting expression. How would you reconstruct the modulating signal from here?
2. Fourier transform the square of the amplitude modulated signal (i.e. \( \text{sig}(t)^2 \)), and visualise.
3. Zoom in to view the frequency range 0 Hz to 300 Hz and identify the components that are retained by a low pass filter designed to cut-off at 100 Hz. Create a filter consisting of zeros and ones and use it to generated the filtered data.

4. Inverse Fourier transform the filtered data and take the square root. The demodulated signal is reconstructed after subtracting the average signal. Compare your demodulated signal with the original modulation.
11 Quantum mechanics (week 8)

This study involves working with a pre-prepared Mathematica notebook written in the style of a small study. The notebook structure reflects that which we expect you to use in answering the Homework in Week 10. The notebook has four sections: 1) a brief introduction, 2) a step-by-step commented code that undertakes the study, calculates the results and display them, 3) a short discussion/conclusion section, and 4) a reference section containing any references to other material used. Additionally, to exercise your debugging skills, the code in the pre-prepared notebook has a number of bugs and errors and is not initially fully-functional.

11.1 Schrödinger’s equation in 1D

The time-dependent Schrödinger equation is:

\[
i\hbar \frac{\partial}{\partial t} \Psi(x, t) = -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \Psi(x, t) + V(x)\Psi(x, t) = H\Psi(x, t)
\]  

(11.1)

where \(m\) is the particle’s mass, \(V(x)\) is the potential field experienced by the particle, and \(\Psi(x, t)\) is the particle wave function.

The quantum eigenstates correspond to solutions having a constant energy that take the form

\[\Psi(x, t) = e^{-iE_n t/\hbar} \psi_n(x)\], where \(\psi_n(x)\) is the wave function having the energy \(E_n\). This stationary wave is a solution of the time-independent Schrödinger equation:

\[E_n \psi_n(x) = -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial x^2} \psi_n(x) + V(x)\psi_n(x).
\]  

(11.2)

More generally, a particle’s wave function can be written as a superposition of stationary eigenstates:

\[\Psi(x, t) = \sum_n a_n e^{-iE_n t/\hbar} \psi_n(x)
\]  

(11.3)

where \(a_n\) defines the amplitude of each of the states. This superposition allows us to define the probability density \(\rho(x, t) = |\Psi(x, t)|^2\) which represents the probability of finding the particle at position \(x\) at time \(t\).

1. Open the Schrödinger study notebook using the command:

   NotebookOpen[path <> "notebook/Schrodinger-B.nb"]

2. Debug the code so that it works as intended. The aim of each line of the code is explained in the related comments inside the code. The best method to debugging is to separate the code into smaller parts, probably a line at a time, and check that they give you the expected output. If not then break the line down further into its functional chunks and check these individually. Make sure to correct one line before moving on to the next as errors earlier can cause related errors later, even though the later code is syntactically correct.

3. Change the potential of the well to introduce a slope at the bottom of the well. Study the change of the ground state as a function of the gradient of the slope.

4. Observe the influence of the gradient of the slope on the other bound states. Think about which levels are most affected and why?

11.2 Study

5. Consider two potential wells close to each other. Using a 50:50 superposition of the first two bound states, visualise the effect of tunnelling between the two wells for two different well-separation distances. Discuss your findings in a short paragraph and ask a demonstrator to confirm your understanding.
6. Considering the analytic expression of the probability density of the superposition, determine the frequency of the tunnelling oscillation. Use ComplexExpand to expands the expression of the superposition assuming that all parameters are real.

7. Create a table of tunnelling oscillation values for a range of well-separation distances ranging from 0.1 to 1 (in the Natural Units used). Fit these data points with an exponential.

11.3 Advanced problem

Study the effect on tunnelling of changing the barrier height between the two wells.

You can also check the numerical solutions of Schrödinger’s equation in the case of a harmonic potential against the analytic solutions.

\[
\psi_n(x) = \frac{1}{\sqrt{2^n n!}} \cdot \left(\frac{m\omega}{\pi\hbar}\right)^{1/4} \cdot e^{-m\omega x^2/2\hbar} \cdot H_n \left(\frac{m\omega}{\hbar} x\right), \quad n = 0, 1, 2, \ldots
\]

where \( H_n \) are the Hermite polynomials and the potential is defined by \( V(x) = (m/2)\omega^2x^2 \). You might also want to compare the numerical solution calculated using SI units with that using Natural Units.
12 Geometric optics (week 9)

12.1 Physics background

In this study we will model the propagation of rays through simple optical systems consisting of lenses and curved mirrors. To do this, we use the small angle approximation, i.e. the rays propagate in a paraxial regime (close to the optical axis), where the following relationship is valid:

\[
\begin{pmatrix}
  x_2 \\
  x'_2
\end{pmatrix}
= \begin{pmatrix}
  A & B \\
  C & D
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  x'_1
\end{pmatrix}
\]

In this, \( x_1 \) and \( x'_1 \) are the height and gradient of the incident ray with respect to the optical axis at position \( z_1 \), and \( x_2 \) and \( x'_2 \) of the transmitted ray at position \( z_2 \). The matrix elements \( A \), \( B \), \( C \) and \( D \) describe an optical system consisting of one or multiple optical components. Of particular interest to us are the following fundamental optical components:

- **Distance** \( d \)

- **Thin lens**

- **Curved mirror**

where \( f \) is the focal length of the thin lens, with \( f > 0 \) for a convex lens, and \( R_m \) is the radius of curvature of the mirror with \( R_m > 0 \) for a concave mirror. Propagation through multiple optical components, achieved via matrix multiplication, allows for the modelling of microscopes, telescopes and laser cavities.

12.2 Programming techniques

**Defining rays and optical elements** Initially, we define a function that takes the distance as an argument and generates the ABCD matrix corresponding to a free-space propagation distance.

\[
\text{dist}[d] := \{\{1, d\}, \{0, 1\}\}
\]

MatrixForm[dist[d]]

1. Define functions for the other two fundamental optical components, naming them lens and mirror. Remember that MatrixForm is only used to display matrices and its output cannot be numerically processed.

2. Use the Det function to calculate the determinant of each of the ABCD matrix to show that for each of the optical components we have \( AD - CB = 1 \). This property will be used later.

We further define a single ray, and a single optical element as replacement rules.

\[
\text{ray} = \{z1 \to 0, x \to 2, xprime \to -0.1, z2 \to \infty, \text{wavelength} \to 450\};
\]

\[
\text{oelem} = \{z \to 20, \text{abcd} \to \text{lens}[10], \text{diam} \to 25\};
\]

3. The propagation of a ray through an optical element will change the parameters of the ray. Use a replacement rule \((\text{wavelength} \to \text{a}) \to (\text{wavelength} \to 620)\) to change the wavelength of the ray from blue (450 nm) to red (620 nm). Note the brackets in the replacement rule. These are necessary to clarify which arrow corresponds to the rule.

4. Using the procedure in the previous point, create a list of parallel rays named rays with \( x \) ranging from -5 to 5 in steps of 1. This list corresponds to a bundle of rays.
Predefined functions  To continue with the case study we will use three functions. While there is benefit to typing them in, you might want to access them directly using:

\[\text{NotebookOpen["case/geopt.nb"]}\]

Propagating bundles of rays  The following code defines a function that propagates a list of rays through a single optical element, \(\text{opt}\).

\[
\text{prop1[rays\_, opt\_] := Module[{ray, rep, x1, xprime1, x2, xprime2},}
\]

\[
\text{Flatten[Table[}
\]

\[
\text{rep = Join[ray, opt];}
\]

\[
\text{If[(z2 == Infinity \&\& z1 <= z) /. rep,}
\]

\[
\{x1, xprime1\} = \text{dist[z-z1].\{x, xprime\} /. rep;}
\]

\[
\{x2, xprime2\} = \text{abcd.\{x1, xprime1\} /. rep;}
\]

\[
\text{Join[\{ray\} /. \{(z2 -> a_) -> (z2 -> (z / . rep))\},}
\]

\[
\text{If[(Abs[x1] <= diam/2 \&\& Abs[x2] <= diam/2) /. rep,}
\]

\[
\{ray\} /. \{(z1 -> a_) -> (z1 -> (z / . rep)), (x -> a_) -> (x -> x2),}
\]

\[
(xprime -> a_) -> (xprime -> xprime2), (z2 -> a_) -> (z2 -> Infinity))\}, \{\}],}
\]

\[
\{ray\}, \{ray, rays\}], 1]\]
\]

5. It is essential you understand the above code. What are the roles of the functions \(\text{Join, Flatten}\) and \(\text{Table}\) in this code? Where is the propagation of ray calculated? Go through the code step by step and write short comments explaining these steps.

6. Use \(\text{prop1}\) to calculate the propagation of the bundle of rays defined by the list \(\text{rays}\) through the optical element \(\text{oelem}\). Name these new rays \(\text{prays}\).

Propagate through multiple optical elements  The following code defines a bundle of rays and an imaging telescope consisting of two lenses.

\[
\text{ray = \{z1 -> 0, x -> 2, xprime -> 0, z2 -> Infinity, wavelength -> 450\};}
\]

\[
\text{rays = Table[ray /. \{(xprime -> a_) -> (xprime -> xp)\}, \{xp, -0.01, 0.01, .001\}]};
\]

\[
\text{tele = \{\{z -> 200, abcd -> lens[200], diam -> 25\},}
\]

\[
\{z -> 800, abcd -> lens[400], diam -> 25\}\};
\]

\[
\text{propagateRays[rays\_, optics\_] := Module[\{irays, opt\},}
\]

\[
\text{irays = rays; Do[irays = prop1[irays, opt], \{opt, optics\}; irays}\]
\]

7. Make sure you understand how the \(\text{propagateRays}\) function works.

8. Use \(\text{propagateRays}\) to calculate the propagation of the bundle of rays, defined by the list \(\text{rays}\), through the telescope.

9. Repeat for a bundle of horizontal parallel rays.

Display optical system and combine optical ABCD elements  The following code creates a graphical representation of the rays and optical system.

\[
\text{showOptics[rays\_, optics\_] := Graphics[Join[\{InfiniteLine\{0, 0\}, \{1, 0\}\},}
\]

\[
\text{Table[Blend["VisibleSpectrum", wavelength],}
\]

\[
\text{If[z2 < Infinity,}
\]

\[
\text{Line[\{z1, x\}, \{z2, x + xprime (z2 - z1)\}],}
\]

\[
\{Dashed, HalfLine[\{z1, x\}, \{1, xprime\}]\}/. ray, \{ray, rays\},}
\]

\[
\text{Table[\{HalfLine[\{z, diam/2\}, \{0, 1\}], HalfLine[\{z, -diam/2\}, \{0, -1\}], Dotted, Line[\{z, diam/2\}]\}/. opt, \{opt, optics\}]\]}
\]
10. Check the help file for the different commands used in the definition of showOptics and make sure you understand how it works. Comment the code.

11. Use showOptics to display the propagation of parallel rays and rays from a point source through the telescope. You can use Show together with AspectRatio and PlotRange to adjust the display range.

12. Use matrix multiplication to calculate the ABCD matrix for the whole telescope. Make sure you multiply the ABCD matrices in the correct order. This telescope configuration is called a 4F arrangement.

13. A generic 2x2 array can be defined by Array[p, {2, 2}]. Show that the determinant of the matrix product of two different generic matrices is equal to the product of their individual determinants i.e. det(P.Q) = det(P) * det(Q).

14. Deduce, with reference to point 2 above, that in general the determinant of any ABCD matrix is equal to one.

12.3 Study

We want to study the properties of the optical system as a function of its constituent components. We will look at magnification factor, energy conservation, and at rays in periodic focusing systems.

15. An optical system that images from the object plane to the image plane, both in air, is characterised by the ABCD matrix \[
\begin{pmatrix}
M & 0 \\
C & D
\end{pmatrix}
\], where \(M\) is the magnification factor. Deduce the value of \(D\) in this general case.

16. Deduce the magnification for a 4F optical imaging telescope defined by two lenses with focal length \(f_1\) and \(f_2\) and check against the ones calculated in point 11.

17. For the 4F imaging system, consider an object of height \(h_1\) where each point on the object radiates in a cone angle of \(\Delta \theta_1\). The total radiation is proportional to \(h_1 \Delta \theta_1\). Calculate the total radiation received on the image side, considering that the system is linear, implying that \(h_1\) and \(\Delta \theta_1\) transforms in the same way as \(x\) and \(\theta\). Note that within the paraxial small angle approximation \(\Delta \theta_1\) is equal to the gradient \(x'\) of the ray.

18. Consider a periodically repeating optical system where each period consists of a thin lens of focal length \(f\) and a distance \(d\) of free space propagation. Display the \(x\)-distance of a ray starting with \((x_1, x'_1) = (1, 0)\) as it propagates through the first 100 periods as a function of the period number. Do this for \(f = 10; d = 1\) and for \(f = 1; d = 10\). Explain what you observe.

19. Determine the eigenvalue of the ABCD matrix of a single period and compare the absolute value of these in the two cases. What is the physical meaning of the eigenvalue with respect to the \(x\)-distance of the ray propagating through the system?

20. Use RegionPlot to display the region of stability for our periodically repeating optical system with \(d\) and \(f\) varying respectively between 0 and 10 and between -10 and 10. This is called a "lens equivalent waveguide".

12.4 Advanced problem

1. Consider an optical cavity consisting of two spherical curved mirrors with radii \(R_1\) and \(R_2\) separated by the distance \(L\). The round trip in the cavity corresponds to one period. Calculate the ABCD matrix of one period and plot the region of stability for \(L = 1\) in the parameter space \(R_1\) and \(R_2\) varying each between -4 and 4.

2. The stability region is usually represented as a function of parameters \(g_1 = 1 - L/R_1\) and \(g_2 = 1 - L/R_2\). For \(L = 1\) write \(R_1\) and \(R_2\) as a function of \(g_1\) and \(g_2\) and re-plot the region of stability these parameters varying each between -4 and 4. Check your graph against https://en.wikipedia.org/wiki/Optical_cavity.
13  Waves (week 10)

13.1  Physics background

In this study we will model the propagation of a wave through a chain of springs and masses. The equation of motion of a mass in our spring chain is:

\[ \frac{d^2 u_i}{dt^2} = -k_i(u_i - u_{i-1}) + k_{i+1}(u_{i+1} - u_i) + f_i \]  

(13.1)

where \( u_i \) corresponds to the distance that mass \( m_i \) has moved from its equilibrium position, \( k_i \) is the spring constant of the spring on the left of mass \( m_i \), and \( f_i \) an externally-applied force. In the following, we are considering the first and last spring constants to be zero.

13.2  Programming techniques

**System of equations** Using the Solve function, we can solve systems of equations involving multiple variables written in matrix form. Here, we set up the matrix equivalent of four equations with random integer coefficients and solve for the four unknowns, \( u_1, u_2, u_3 \) and \( u_4 \).

```plaintext
ntotal = 4;
b = RandomInteger[20, ntotal];
var = Array[u, ntotal]; (*vector of mass displacements*)
mat = RandomInteger[20, {ntotal, ntotal}];
(*The following line visualises the matrix equation to be solved*)
MatrixForm[mat] MatrixForm[var] == MatrixForm[b]
```

1. Use Table to create a list of four equations from matrix mat.
2. Display the system of equations using TableForm. Note that the outputs from TableForm and MatrixForm are for visualisation purposes only and should not be used in subsequent calculations.
3. Use Solve to solve the equations.
4. Increase the number (ntotal) of unknowns to 10, and repeat the steps above.

**Building the band matrix** The forces acting on the masses in the chain of springs can also be written in the form of a matrix involving spring constants applied to a vector of mass displacements, \( u_i \). For 10 masses (11 springs) we have:

```plaintext
ntotal = 10;
var = Array[u, ntotal]; (*vector of mass displacements*)
kd = Array[k, ntotal + 1]; (*list of spring constants*)
kd[[1]] = kd[[-1]] = 0;
kmatrix = Table[Which[
i == j, -kd[[i]] - kd[[i + 1]],
i == j + 1, kd[[i]],
i == j - 1, kd[[i + 1]],
True, 0], {i, ntotal}, {j, ntotal}];
MatrixForm[kmatrix] MatrixForm[var]
```

5. Use the dot product between the spring-constant matrix, kmatrix, and the vector of displacements, var, to create a list of the forces acting on each mass. Display these using TableForm and compare with equation (13.1) in the "Physics background" section.
6. Redefine the list of spring constants, kd, using the ConstantArray function with a uniform spring constant of 25N/m. Display the, now numerical, spring-constant matrix kmatrix using MatrixForm.
7. Change the list of spring constants such that the first five springs have a spring constant of 25N/m and the remainder 100N/m. Recalculate and re-display using MatrixForm.

8. Increase the number of masses to 200, and go back to using a uniform spring constant of 25N/m. Display the numerical spring-constant matrix using the MatrixPlot function.

**Solving the equations** In order to solve the equations of motion for the spring chain, we need to define the inertial masses and the externally-applied force. We'll assume that each of the masses have \( m_0 = 1.2 \) kg, and are initially at rest.

\[
t_{\text{max}} = 25; \quad m_0 = 1.2; \\
u_0 = \text{ConstantArray}[0, \text{ntotal}]; \quad u_{p0} = \text{ConstantArray}[0, \text{ntotal}]; \\
p[t_] := \text{Exp}[-(t - 6)^2/4] \cos[\pi/2 (t - 6)]; \\
b_1 = \text{ConstantArray}[0, \text{ntotal}]; \quad b_1[[1]] = 1; \\
fi = \sqrt{\text{springk0} \times m_0} \times \frac{\text{D}[p[t], t]}{b_1};
\]

\[
solv1 = \text{NDSolve}[[m0 \times u''[t] == \text{kmatrix}.u[t] + \text{fi}, u[0] == u_0, \quad u'[0] == u_{p0}], u, \{t, 0, t_{\text{max}}\}][[1]]; \\
\text{Animate}[\text{ListLinePlot}[u[t] / solv1, \text{PlotRange} \to \{-1.5, 1.5\}, \text{Frame} \to \text{True}, \\
\quad \text{FrameLabel} \to \{"mass chain position (i)"}, "\text{displacement}"\}], \{t, 0, t_{\text{max}}\}, \\
\quad \text{AnimationRepetitions} \to 1]
\]

9. Build a list of the displacements of the 200 masses for \( t \) varying between 0s and 25s in 50 values.

10. Display these displacements using ListPlot3D and ListDensityPlot to visualise the time-space propagation of the pulse. Scale and label the axes appropriately. Be aware that Mathematica sometimes zooms in, so the plot might not initially show what you expect if you have not set an appropriate PlotRange.

13.3 Study

We want to study the properties of the wave, its behaviour both at the end of the spring chain, and when it encounters a step change in the spring constant.

11. Considering a chain of identical springs, we want to determine the phase velocity (“wave speed”) as a function of the spring constant. Firstly, solve for a spring constant of 25N/m and build a list of the position of the maximum displacement value as a function of time.

12. Fit the linear section of the above list with a linear dependence and determine the slope (phase velocity).

13. Repeat the above to determine the phase velocities for a few different values of the spring constant in the range from 25N/m to 100N/m. Verify that the phase velocity is given by \( v = \sqrt{k/m} \) where \( k \) and \( m \) are the uniform spring constant and the mass, respectively.

14. Increase \( t_{\text{max}} \) to 100 s with a spring constant of 100N/m. Observe the behaviour of the wave as it reaches the end of the chain. Note, that we have a open-end boundary condition as the last spring is implicitly non-existent, i.e. \( k_{201} = 0 \).

15. Change the boundary condition to a closed boundary condition by introducing on the last mass an additional force equal to \(-k_{200}u_{200}(t)\). This is equivalent to adding a spring connected to a fixed point. On the same graph, visualise the reflections of a wave for both an open and a closed boundary condition.

16. Waves also reflect as a result of a change in phase velocity, e.g. from a change in propagation medium. Consider the case where the first 100 springs have a spring constant of 100N/m and the rest 49N/m. Display the motion of the wave for the first 25 s and determine the transmission and reflection coefficients of the wave amplitudes by measuring their peaks.
17. Verify the reflection coefficients against \( t = 2v_1/(v_1 + v_2) \) and \( r = (v_1 - v_2)/(v_1 + v_2) \) where \( v_1 \) and \( v_2 \) are, respectively, the phase velocities in the first and the second medium. These are Fresnel-like equations written as a function of the phase velocities in the two media.

18. Double check your simulation against the Fresnel-like formulas when inverting the two spring constants, i.e. 49N/m followed by 100N/m.
14 Waves in periodic structures (week 11)

14.1 Physics background

In this study we will model the propagation of a wave through a chain of springs and masses that have a periodic variation of their properties. This will allow us to study propagation stop-bands and resonant cavities. To enable this study, we need to introduce non-reflecting boundary conditions for the spring chain. We do this by applying a force \( b_n \) to the last mass

\[
m \frac{d^2 u_n}{dt^2} = -k_n (u_n - u_{n-1}) + b_n
\]

where \( n \) is the last mass and \( k_{n+1} = 0 \).

We want to determine an expression for \( b_n \) such that a wave travelling towards this boundary is not reflected. To do this we need to account for the time it takes to travel from mass \((n-1)\) to mass \( n \). These two masses are one unit-length apart and the wave travels with velocity \( \sqrt{\frac{k}{m}} \). This time interval is given by \( \sqrt{\frac{m}{k}} \) which lead to the following:

\[
u_{n-1}(t) = u_n(t) + \sqrt{\frac{m}{k}} u'_n(t) \]

\[
u_{n-1}(t) = u_n(t + \sqrt{\frac{m}{k}}) \] (14.2)

where we have used a first order Taylor expansion to express the displacement of mass \((n-1)\) as a function of the displacement of the mass \( n \). We now substitute the expression \((14.3)\) into \((14.1)\) and solve for \( b_n \) such that the resultant force is zero. This happens for \( b_n = -k u'_n \sqrt{\frac{m}{k}} \).

This definition of the non-reflecting boundary condition is valid provided that the time delay \( \sqrt{\frac{m}{k}} \) between successive masses is small, i.e. when the number of masses is increased while keeping constant the total mass, spring constant and length of the chain. To do this, we define the total mass of the chain \( m_T \), its total length \( L_T \), its overall spring constant \( k_T \) and the number of masses \( n_T \). Leading to:

\[
m = \frac{m_T}{n_T} \quad ; \quad k = k_T n_T \quad ; \quad h = \frac{L_T}{n_T} \quad ; \quad v = \frac{\sqrt{k}}{m} = L_T \sqrt{\frac{k_T}{m_T}} \] (14.4)

where \( h \) is the equilibrium distance between two successive masses and \( v \) the wave phase velocity. This velocity is the same as we saw in last week’s Waves study, with \( h = 1 \). To decrease the time delay \( \sqrt{\frac{m}{k}} \) we need to increase \( n_T \).

14.2 Programming techniques

**Non-reflecting boundaries** Non-reflecting boundary conditions rely on calculating a boundary force which cancels out the incident wave. Initially, we determine this force using the \texttt{Series} function which calculates the Taylor expansion of any function.

\[
\text{unm1} = \text{Normal}[\text{Series}[\text{un}[t + dt], \{dt, 0, 1\}]];
\text{force} = -k (\text{un}[t] - \text{unm1}) + \text{gamma} \text{ un'}[t] /. \text{dt} \rightarrow \text{Sqrt}[m/k];
\text{Simplify}[\text{force}]
\]

1. Evaluate the Taylor expansion function (\texttt{Series}) without the \texttt{Normal} function and observe that it has an \( O(dt^2) \) term that represents an error term of the order of \( dt^2 \).
2. Use \texttt{Solve} to determine the coefficient \( \text{gamma} \) such that the resultant force on the last mass is zero.
3. Modify the closed-boundary code from the waves case study to change the right hand side boundary to be non-reflecting. Use \( p(t) = e^{x(t-6)^2/4} \cdot \cos(\pi(t-6)/2) \) for the left-hand side input. Check visually the effects from this boundary. You will notice a small back reflection. Note, it is possible to further reduce this back reflection by using higher order terms in the Taylor expansion.
4. Modify the left hand side boundary to non-reflecting boundary and check the input amplitude of the input pulse. Why is it now half of its previous value? Correct the input by a factor of 2.
Increasing number of masses Using the definitions in the background section we can increase the number of masses without changing the propagation properties of the chain.

\[ \text{ntotal} = 100; \quad (* \text{number of masses}*) \]
\[ \text{Ltotal} = 20; \quad (* \text{total length}*) \]
\[ \text{mtotal} = 200 \quad (* \text{total mass}*) \]
\[ \text{ktotal} = 1; \quad (* \text{overall spring constant}*) \]
\[ \text{m0} = \frac{\text{mtotal}}{\text{ntotal}}; \quad (* \text{individual masses}*) \]
\[ \text{springk0} = \frac{\text{ktotal} \times \text{ntotal}}{\text{ntotal}}; \quad (* \text{individual spring constants}*) \]
\[ \text{h} = \frac{\text{Ltotal}}{\text{ntotal}}; \quad (* \text{rest distance between masses}*) \]
\[ \text{v0} = \frac{\text{h}}{\sqrt{\text{springk0} / \text{m0}}}; \quad (* \text{wave phase velocity}*) \]

5. Use PowerExpand to check analytically that the velocity of the pulse does not change when changing the number of masses.

6. Verify that the amplitude of the reflected wave decreases when increasing the number masses from 100 to 200.

7. Change the spring constant for \(8 < x < 12\) such that the phase velocity is \(n = 2.5\) times smaller than \(v0\) outside this region. Use ListDensityPlot to show the time-space propagation of the pulse.

8. Plot function \(f(t) = \frac{(1 + \text{erf}(t - 6))}{2}\) where erf() is the error function. This function can be used to achieves a smooth switch to a constant input. Change the input to \(p(t) = f(t) \exp(i \times \pi t/2)\) corresponding to the switch-on of a continuous monochromatic input. Propagate until a steady state is achieved and visualise the absolute value of the displacement at the input and output. Determine the intensity transmission and reflection coefficients in the steady state case making sure you subtract the incident wave from the total field when calculating the reflection coefficient.

9. Change the thickness of the region where \(n = 2.5\) such that it is one wavelength in thickness and re-display the absolute value of the displacement at the input and output. Recalculate the intensity transmission and reflection coefficients.

Single frequency To correctly calculate the transmission and reflection coefficients of complex structures it is important to increase the number of masses considered. In this case, it makes sense to write the equations of motion of the masses in a vector form:

\[ m \frac{d^2}{dt^2} \mathbf{U}(t) = \mathbf{K} \cdot \mathbf{U}(t) + \mathbf{B} \cdot \mathbf{U}(t) + \mathbf{F}(t) \quad (14.5) \]

where \(\mathbf{U}(t)\) is a vector with each component corresponding to the displacement of one mass and where \(\mathbf{F}(t)\) is an external force. The matrices \(\mathbf{K}\) and \(\mathbf{B}\) are constant matrices describing the spring constants and the boundary conditions.

However, solving this vector system is computationally expensive. To overcome this difficulty, we consider directly the stationary case at a single input frequency (monochromatic). In this case, we note that all masses oscillate at the same frequency of the form \(\mathbf{U}(t) = u_j(t) = a_j \exp(i \omega t) = \mathbf{A} \exp(i \omega t)\), where \(\omega\) is the oscillation frequency and \(a_j\) is the complex amplitude of oscillation of mass \(j\). These amplitudes are gathered together in vector \(\mathbf{A}\).

In this case, the vector system of equations (14.5) becomes:

\[ -m \omega^2 \mathbf{A} = \mathbf{K} \cdot \mathbf{A} + \mathbf{B} \cdot \mathbf{A} + \mathbf{F}_\omega \quad (14.6) \]

where \(\mathbf{F}_\omega\) corresponds to the external forces at the frequency \(\omega\).

This approach is implemented in the following code:

\[ \mathbf{A} = \text{Array}[a, \text{ntotal}]; \]
\[ \omega = 2 \text{ Pi freq IdentityMatrix}[\text{ntotal}]; \]
bnc = Sqrt[springk0 m0] (bc1.w) + Sqrt[springk0 m0] (bc2.w);
eqm = m0 w.w - kmatrix - bnc;
fi = b1;
solf = Solve[{eqm.A == fi}, A][[1]];
ListLinePlot[Abs[A]^2 /. solf, PlotRange -> All , Frame -> True,
FrameLabel -> {"Position in optical system", "Intensity"},
PlotLabel -> "Intensity through the optical system"]

10. Change the input fi to calculate the stationary case and determine the reflection and transmission coefficients. Compare to the these values to the one calculated previously.

11. Create two layers, the first one with an velocity index $n_1 = 1.5$ and the second $n_2 = 2.5$. Make the thicknesses of these layers a quarter of the wavelength taking into account the velocity index $d_1 = \lambda/(4 \times n_1)$ and $d_2 = \lambda/(4 \times n_2)$. Determine the reflection and transmission coefficients.

14.3 Study

We want to study the reflection of waves from a Distributed Bragg Reflector (DBR) and the design of resonant cavities. These structures are used in optics for laser cavities.

12. Define a list of the index of refraction ($n$) consisting of 10 periods, each period consisting of two quarter-wave layers (use $\lambda = 5$). The material of each quarter-wave layer has an index of refraction of 1.5 and 2.5, (in optics this would correspond to $SiO_2$ and $TiO_2$). Note that the wavelength is linked to the frequency as $\lambda = v_0/f$ where $f$ is the frequency. The functions CenterArray and Join might be useful.

13. Solve the single frequency equations for $\lambda = 5$ and $\lambda = 7$ (while keeping the quarter-wave layers for $\lambda = 5$) and determine the transmission and reflection coefficients for these two cases.

14. Create a list of reflection and transmission coefficients for $\lambda$ varying from 3 to 8 in steps of 0.02. Plot this list.

15. Check that the total intensity is conserved.

16. Split the structure in the middle and introduce a $\lambda/4$ gap with $\lambda = 5$. Solve the propagation of a wave for $\lambda = 5$ and $\lambda = 5.5$. What do you observe and what does it mean?

17. Plot the transmission and reflection coefficients as a function of $\lambda$ varying from 3 to 8 in steps of 0.02.

18. Zoom in on the cavity resonance line by considering $\lambda$ varying from 4.9 to 5.1 in steps of 0.001.

19. Redo steps (16) and (17) with the gap set to $17\lambda/4$. You will need to increase the total length of the spring chain and its total mass to accommodate the longer structure. What do you observe and what does it mean?

20. Redo steps (16) and (17) after decreasing the number of periods on each side of the gap in the DBR stacks. What do you observe and what does it mean?

14.4 Advanced problem

An other approach to numerically modelling waves relies on Finite-difference time-domain method (FDTD). This approach is suitable for larger simulations in 2 and 3 spatial dimensions. The first chapter of the e-book [1] explains this approach for the 1D case and gives a simple C++ implementation. Using the explanations in the chapter, the Rosetta code website and the C++ code, translate this FDTD code into Mathematica.

15  Past paper (2018)

1. Define the function \( f(x, y) \) and the parametric curves \( (x_1(\theta), y_1(\theta)) \) and \( (x_2(\theta), y_2(\theta)) \):

\[
\begin{align*}
  f(x, y) &= 15 \sin^3(x + 1) + \cos(2y), \\
  x_1(\theta) &= \cos(\theta + 2), \\
  y_1(\theta) &= \sin(\theta), \\
  x_2(\theta) &= \cos(\theta) + 0.6, \\
  y_2(\theta) &= \sin^2(\theta).
\end{align*}
\]

Define the functions \( g \) and \( h \):

\[
\begin{align*}
  g(\theta) &= f(x_1(\theta), y_1(\theta)), \\
  h(\theta) &= f(x_2(\theta), y_2(\theta)).
\end{align*}
\]

Plot \( g(\theta) \) and \( h(\theta) \) together for \( \theta \) in the range \(-\pi\) to \( \pi \) radians, determine the \( \theta \)-values of the four crossing points. Re-plot the functions for \( \theta \) in the range delimited by the left-most and right-most crossing points. Use the Filling plot option to shade, in red, the area between the two curves. Determine the area of the shaded region.

Submit the code necessary to plot, shade, and calculate the crossing values and area.

2. Use the command:

\`\`Import["https://www.st-andrews.ac.uk/physics/ph3080/data/fourier18b.dat"]\`

to import data consisting of two columns representing a signal measured at regular time intervals. The data file has header information on the first two lines. Plot separately the signal and the absolute value of its Fourier transform in the frequency range

\[
0 \text{ Hz} \leq f \leq 100 \text{ Hz},
\]

remembering that the frequency step size in the Fourier list is given by \( \Delta f = 1/T \), where \( T \) is the overall sampling duration. Determine the frequency of the peak with the highest amplitude in this range.

Submit the code necessary to import and plot the data, plot the Fourier transform, and find the frequency.

3. Use the

\`\`NotebookOpen["https://www.st-andrews.ac.uk/physics/ph3080/code/soliton.nb"]\`

command to open a Mathematica notebook containing bugs and mistakes. The code in this notebook is an attempt to study the interaction between two water-wave pulses. The larger pulse catches-up, passes through, and overtakes the smaller one. These particular pulses are referred to as solitons.

The imported code does not work as intended, and evaluation will result in various error messages. Your task is to find and correct all the errors, and make the code fully functional. The aim of each piece of code is explained in the related comments inside the code. You need to check every line to ensure it is bug and error free, though some lines already are correct. The best method of doing this is to separate the code into smaller parts and check them sequentially.
Submit the debugged code that satisfies all of the comments in the code. The code should not be heavily modified. Your aim is to debug the code as it stands. Each error you correct is associated with a few of the available marks.

4. In the following, we study the motion of a mass, \( m \), attached to one end of a spring that is fixed at the other end to the origin of the coordinate system. The mass is also subject to a constant gravitational field leading to an acceleration, \( g \), in the negative \( y \)-direction.

The position of the mass is defined by \((x(t), y(t))\), which is measured with respect to the fixed end of the spring. The equilibrium length of the spring is \( L = 1 \) m and its spring constant is \( k \). Newton’s equations of motion for this system are:

\[
\begin{align*}
    m \frac{d^2 x}{dt^2} &= -k(r - L) \frac{x}{r}, \\
    m \frac{d^2 y}{dt^2} &= -k(r - L) \frac{y}{r} - mg ,
\end{align*}
\]

where \( m = 1 \) kg, \( g = 9.81 \) m/s\(^2\), and \( r = \sqrt{x^2 + y^2} \) correspond to the mass, gravitational constant, and the length of the spring, respectively.

Using \( k = 50 \) N/m, solve for the motion of the mass, assuming that initially the mass is at the position \((x(0) = 0 \text{ m and } y(0) = -1 \text{ m})\) and has the initial velocity \((v_x(0) = 1 \text{ m/s and } v_y(0) = 0 \text{ m/s})\). Plot the trajectory of the mass for the time, \( t \), ranging from 0 s to 10 s. Verify graphically that the total energy \( E_T \) is conserved, with the value equal to 0.5 J, where

\[
E_T = \frac{k}{2}(r - L)^2 + \frac{m}{2}(v_x^2 + v_y^2) + mg(y + L) ,
\]

and where the potential energy is measured with respect to \( y = -L \).

Submit the code necessary to solve the differential equations, to define the energy, and to create the plots.

5. Use the NotebookOpen["https://www.st-andrews.ac.uk/physics/ph3080/code/magneticbobgray.nb"] command to import a Mathematica notebook containing a model of a system.

In the following, we study the motion of an iron bob attached to a string hanging over three magnets. The position of the bob in the \( x - y \) plane is defined by \((x(t), y(t))\). The equations describing its motion are:

\[
\begin{align*}
    m \frac{d^2 x}{dt^2} &= -kx - \mu \frac{dx}{dt} + c \left( \frac{x_1 - x}{r_1^3} + \frac{x_2 - x}{r_2^3} + \frac{x_3 - x}{r_3^3} \right) , \\
    m \frac{d^2 y}{dt^2} &= -ky - \mu \frac{dy}{dt} + c \left( \frac{y_1 - y}{r_1^3} + \frac{y_2 - y}{r_2^3} + \frac{y_3 - y}{r_3^3} \right) , \\
    r_i &= \sqrt{h^2 + (x_i - x)^2 + (y_i - y)^2} ,
\end{align*}
\]
where \( m = 1 \text{ kg} \), \( k = 0.2 \text{ N/m} \), \( \mu = 0.1 \text{ Ns/m} \), and \( c = 1 \text{ Nm}^2/\text{s} \) correspond to the mass, the pendulum constant, the friction coefficient, and the magnetic strength coefficient, respectively. The positions of the three magnets is given by \((x_1, y_1) = (\sqrt{3}, 1) \text{ m}\), \((x_2, y_2) = (-\sqrt{3}, 1) \text{ m}\), and \((x_3, y_3) = (0, -2) \text{ m}\), and \( h = 0.25 \text{ m} \) corresponds to the height of the pendulum above the \( x-y \) plane when at rest in the absence of the magnets.

(a) Using the trajectory of the bob calculated in the imported notebook question, determine the magnet closest to the bob at \( t = 25 \text{ s} \). Define a function that outputs the number (1, 2, or 3) of the closest magnet at \( t = 25 \text{ s} \), the function should take the initial position \((x_0, y_0)\) and friction coefficient \( \mu \) as arguments. You should assume that the bob is initially at rest. Check the output of this function with the trajectory calculated earlier.

Submit the code necessary to define the function and check its output.

(b) For a bob initially at rest with \( \mu = 0.1 \text{ Ns/m} \), vary the initial position of the bob on a raster with \( x(0) \) and \( y(0) \) ranging from -3 m to 3 m in 50 values. Determine the closest magnet at \( t = 25 \text{ s} \), using the function defined in part (a), or otherwise. Create a grayscale raster plot with these closest magnet positions, where magnet 1 is represented in black, 2 in gray, and 3 in white. Repeat the grayscale raster plot for \( \mu = 0 \text{ Ns/m} \). Analyse and comment on the physical meaning of these two grayscale maps.

Submit the code used to create the grayscale maps and a short paragraph, as a Mathematica comment, outlining your observations and analysis.

(c) With \( \mu = 0 \text{ Ns/m} \), \( x(0) = 2 \text{ m} \), and \( y(0) = -0.5 \text{ m} \), define the function \( E_k(t) = \frac{1}{2}mv^2 \) that calculates the kinetic energy of the bob as a function of time. Using \texttt{ParametricPlot3D}, plot curve \((x(t), y(t), E_k(t))\) for the time, \( t \), ranging from 0 s to 2000 s. Increase the number of plot points to 300 to ensure the trajectory is properly represented. Observe the shape of this trajectory, and comment on the physical meaning of its shape. Your analysis and conclusions might be evidenced by further code.

Submit the code used in your study including a short paragraph, as a Mathematica comment, outlining your observations, analysis, and conclusion.