Tools for Teaching Computational Mathematics

Matthew Paul Skerritt
B.CompSc, B.Math(Hons)

Discipline of Mathematics, University of Newcastle
Callaghan, NSW, 2308, Australia

Thesis submitted for the degree of
Masters of Philosophy (Mathematics)
March 2014

Statement of Originality

Signature: .... Date: ....

Thesis by Publication

I hereby certify that this thesis is in the form of a pair of published books of which I am a joint author. I have included as part of the thesis a written statement from each co-author, endorsed by the Faculty Assistant Dean (Research Training), attesting to my contribution to the joint publications.

Signature: .... Date: ....

Statement of Contribution

I, Jonathan M. Borwein, attest that Master of Philosophy (Mathematics) candidate Matthew P. Skerritt was the principal author of the publications entitled “An Introduction to Modern Mathematical Computing with *Maple*” and “An Introduction to Modern Mathematical Computing with *Mathematica*” and wrote the full draft of both texts.

Signature: .... Date: ....

Laureate Professor Jonathan M. Borwein

Signature: .... Date: ....

Matthew P Skerritt

Signature: .... Date: ....

Associate Professor Jenny Cameron
Assistant Dean Research and Training

Abstract

So called “computer algebra” or “symbolic computation” systems such as Maple, and Mathematica have become complete mathematical computation workspaces with a large and constantly expanding built-in “knowledge base”. They aim to provide exact mathematical answers to mathematical questions, and have opened the way for so called “experimental” computer-assisted mathematics in both the pure and applied fields. Furthermore, it has been only recently (perhaps the last 5 to 10 years or so) that personal computers have been quick enough that one might feasibly experiment with mathematics applicable to a second year student at a cognitively satisfying speed for the learner. The author has designed a second year mathematics course to introduce students to computer algebra systems, and has written textbooks to go with the course: “An Introduction to Mathematical Computing with Maple”[1], and “An Introduction to Mathematical Computing with Mathematica”[2]. We discuss the process of creation of both the course and the books, and the differences between our approach, and prior approaches.

[1]ISBN: 978-1-4614-0121-6, URL: `http://www.springer.com/mathematics/book/978-1-4614-0121-6`
[2]ISBN: 978-1-4614-4252-3, URL: `http://www.springer.com/mathematics/book/978-1-4614-4252-3`

Table of Contents

CHAPTER 1

Introduction

"An Introduction to Modern Mathematical Computing with *Maple*" was published in July 2011 and "An Introduction to Modern Mathematical Computing with *Mathematica*" was published in August 2012. The author has designed a second year mathematics course to introduce students to computer algebra systems, and has written the aforementioned textbooks to complement this course. The texts constitute the thesis proper. The purpose of this document is to provide context to the creation of the texts and the course, as well as to detail the decision processes that were part of their creation.

1. Notation

For the duration of this document, "An Introduction to Modern Mathematical Computing with *Maple*" will be referred to as *the Maple text*, and similarly "An Introduction to Modern Mathematical Computing with *Mathematica*" as *the Mathematica text.* Collectively they will be referred to as *the texts*, and individually as *the text* when the context is unambiguous. When referring to sections or subsections of the text, the § symbol will be used (e.g., §1.1.3 of the *Maple* text). When referring to subsections or sections within this document, the word "Section" will be used (e.g., Section 3.2)

2. Background

So-called "computer algebra" or "symbolic computation" systems such as *Maple*, and *Mathematica* have become complete mathematical computation workspaces with a large and constantly expanding built-in "knowledge base". They aim to provide exact mathematical answers to mathematical questions, and have opened the way for so-called "experimental" computer-assisted mathematics in both the pure and applied fields.

Such systems are not especially new; *Maple* dates back to 1980, and *Mathematica* was first released in 1988. However, it has been only in the last decade that personal computers have become sufficiently powerful that a learner might feasibly experiment with mathematics applicable to a second year student at a satisfying speed.

The University of Newcastle introduced a mathematical software course shortly before the year 2000.[1] The course was named "Mathematical Software" and had the course code MATH2600. It consisted of six weeks introducing students to LaTeX, and six weeks introducing students to *Maple.*

In 2008 Professor Jon Borwein, having recently taken a position at the University of Newcastle, set about updating the course to better reflect the use of computer algebra in

[1]The exact number of years is hard to pin down, as the beginning of the course predates widespread use of computerised record keeping within the Mathematics department.

modern mathematics. He found that no suitable reference material existed. Consequently the author was given the tasks of re-designing the course, and producing course notes which would later be turned into a textbook. The process was iterative over several years, simultaneous with teaching the course.

The brief for the updated course was for it to cover four broad topics: Number Theory, Calculus, Linear Algebra, and Visualisation and Geometry. Exploration and experimentation were to be important aspects of the new course. The remainder of the details were left to the author, who chose to use familiar first-year mathematics to introduce students to computer-assisted mathematics. An assessment schedule was chosen consisting of a two-hour laboratory test at the end of each topic, as well as a final assignment in which students explore a new (to the student) mathematical topic of their choice.

An important consideration was to teach mathematics primarily and computer algebra tools secondarily. The computer algebra system was treated as a tool whose use was to a greater end: to explore and learn mathematics. This philosophy is in keeping with Borwein's own work in the areas of experimental mathematics [**2, 3, 4, 5**].

Although the updated course was originally intended to introduce students to experimental and computer-assisted mathematics, the teaching of the course has suggested that computer algebra systems may very well be an effective tool for students in learning and understanding more traditional mathematics.

3. Existing Literature

The following texts were evaluated for the course:

"The Maple V Primer" and "The *Maple* Book" both by Frank Garvan [**7, 8**] are good introductions and guides to *Maple*. They were unsuitable primarily because they describe a significantly older version of *Maple* than was being used in the course (version 5 in the books and version 12 in the course). Of note is that [**8**] works through mathematical topics in the order usually experienced by a student not using a CAS: starting with High School Algebra, then moving through Calculus, Differential Equations, Linear Algebra, Multivariable and Vector Calculus, Complex Analysis, Special Functions, and Statistics. Interspersed between these topics, where appropriate, are chapters dealing with *Maple* specific notions such as Data Types, Graphics, Programming, and the like.

"Introduction to Maple" by André Heck [**10**] is an excellent resource, but has two properties that make it unsuitable as a course text. Firstly, it is written for a version of *Maple* that is too old (specifically, version 8).[2] Secondly, its main aim is to teach *Maple* whereas the primary goal of our course is to teach mathematics. It should be noted, however, that the means with which it teaches *Maple* involves good and interesting mathematical examples. Despite its age, it is a highly effective text and covers the technical particulars of *Maple* in far greater detail than the texts that form the basis of this thesis.

[2]We should note that at the time of writing this document version 18 of *Maple* has only just been released, however the author has not yet had a chance to use it. Even though the *Maple* text was written for version 12 of the software, no changes have been introduced in the intervening versions (up to version 17) which would require any significant change in the content of the text.

It is an excellent supplemental text for students or readers who wish to know more about *Maple* itself and is highly recommended by the author.

In the book "Solving Problems in Scientific Computing Using *Maple* and MATLAB" by Gander and Hřebíček [**6**] each chapter is dedicated to a single problem which is presented mathematically and explored using *Maple* or MATLAB. However, the book presents topics more advanced than we teach in MATH2600 and makes no effort to introduce *Maple*.[3] As such, it was unsuitable for the course.

"A Short Course in Mathematical Methods with *Maple*" by Henrik Aratyn and Constantin Rasinariu [**1**] also presents topics too advanced for use in the course as planned. It is written for *Maple* version 10, which is older than the version being used at the time of the course update, but the book is still recent enough to be serviceable. The book also assumes, for the most part, existing familiarity with *Maple*, and provides only an appendix by way introduction for the reader not already familiar. What is remarkable about this book is the way it presents the material. The first two thirds of the book, consisting of seven chapters, is a typical mathematical treatise on the topics in question. The final third of the book, consisting also of seven chapters, covers the same topic as the original seven chapters (in the same order), this time using *Maple*. It would be a good book for students to graduate to using after completing the course.

"Geometry of Curves and Surfaces with *Maple*" by Vladimir Rovenski [**11**] is too specific in its topic area to be considered as a text for the MATH2600 course. Additionally, some of its topics are a little too advanced use in the course. It is written in a manner very similar to that adopted by the author for the *Maple* and *Mathematica* texts and would make a good project resource for a strong student's final assignment.

"*Mathematica* in Action" by Stan Wagon [**12**] was unsuitable for the project because it was written for *Mathematica*, not for *Maple*. The aim of the book appears to be primarily to teach *Mathematica*, however it is worth noting because it does so by interspersing unusual or complicated examples throughout the book (among examples more familiar to most readers). Unfortunately, the topics covered do not cover quite as many of the topics the author wished to cover in the updated course. The second edition (cited here, and in the texts as well) is reasonably old—published in 1999—and the third edition was not published until 2010.

"Exploring Mathematics with *Mathematica*" by Theodore W. Gray and Jerry Glynn [**9**] is a remarkable book that is written entirely as a dialogue between the two authors, and occasional guests. However, it was unsuitable for our purposes due to being a book about *Mathematica*, not *Maple*. The writing style allows Gray and Glynn to simultaneously present mathematics primarily, and *Mathematica* secondarily, and to also give a first hand demonstration of computer assisted mathematical exploration and experimentation in action. This book was published in 1991 and references *Mathematica* version 2.[4] Regrettably, it appears that no new versions have been published.

[3]The preface explicitly states that the book is aimed at students already familiar with *Maple*.

[4]For reference, the current version of *Mathematica* is version 9, and version 7 was current at the time the MATH2600 course was first being restructured

4. Supplementary Material

Note that a website for the books is being maintained at `http://carma.newcastle.edu.au/books/mathematicalcomputing/`. At this address can be found: supplementary files containing every piece of *Maple* or *Mathematica* code found in the texts (as worksheet or notebook files respectively), solutions to exercises, errata, and exercise guides. The exercise guide is included in Appendix A, and the reader is encouraged go to the website if they wish to make use of the other resources.

CHAPTER 2

Maple

The *Maple* text began as a set of course notes for the updated MATH2600 course which were written over a semester while teaching the course and, at the end of the semester, were then modified in accordance with the teaching experiences. The course was taught two subsequent times and each time the notes were modified according to the teaching and learning experiences.

As noted previously, the overriding philosophy of both the course and the notes was that the computer environment and code should be subservient to the mathematics. We did not wish to write a "How to use *Maple*" book, nor to teach such a course. *Maple* was very much treated as a means to an end; that end being to learn and explore mathematics. Nonetheless, familiarity with *Maple* and its peculiarities was a necessary by-product of the chosen approach.

The then current version of *Maple* (version 12) used, by default, a "pretty-printed" input scheme known as "2D Input". This input mode automatically typesets superscripts, subscripts, fractions, and the like in a mathematical manner. Versions prior to version 10 used a textual input mode, now referred to as "Maple Input", which could be much more difficult to read for complicated expressions. For example, the equation for the nth Fibonacci number

$$\frac{\sqrt{5}}{5} \cdot \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right)$$

would, in 2D input, look like

$$\frac{sqrt(5)}{5} \cdot \left(\left(\frac{1+sqrt(5)}{2} \right)^n - \left(\frac{1-sqrt(5)}{2} \right)^n \right)$$

and in Maple input would look like

```
(sqrt(5)/5)*(((1+sqrt(5))/2)^n-((1-sqrt(5))/2)^n)
```

We wanted the mathematics on the screen to be as close as possible to mathematics as written on paper within the limitations afforded by prose and ease of input. Note that the 2D input presented above is able to be entered into *Maple* using only the keyboard. *Maple* will automatically move to a superscript position when `^` (caret) is pressed, or to a subscript when `_` (underscore) is pressed, or create a fraction when `/` (forward slash) is pressed. It is possible to input the fully typeset expression—including the radical symbols—by using the mouse to select from a collection of template expressions from a GUI sidebar. However, given the limitations of describing GUI manipulation in prose and the inherent awkwardness of using such a GUI, it was decided to use the subset of the 2D input scheme which could be entirely entered via keyboard.

In order to have the *Maple* code in the text look as similar as possible to that which the student or reader sees on screen, it was required to write our own LaTeX environment for the display of the *Maple* environment. We discovered that an important part of this was the replication of the colours used by *Maple*. A student working with a black-and-white printout of the first draft of course notes expressed confusion regarding the *Maple* code found at the bottom of what is now p72. Specifically, the student had mistaken the output of $plot1 := PLOT(\dots)$ as an input they were expected to enter. This was despite the fact that the output was differentiated by being centred on the page (whereas the input is left-aligned). Adding the appropriate colour to the printout (blue for *Maple* output) has entirely removed this confusion. We were fortunate that the publishers were happy to print the entire book in colour.

Much time was expended in the search for examples which fit within a remarkably thin band of being complicated enough to require (or, at the very least, benefit from) the use of a computer algebra system on the one hand, and yet still be accessible to second year students on the other. The process involved several iterations of teaching the MATH2600 course and seeing first hand how the students dealt with the material. It turns out that some problems that would be difficult on pen and paper alone become accessible to second year students with the aid of the computer (the Collatz conjecture is a clear example).

Similarly, some mathematical concepts beyond the ability of the students to manipulate directly can be introduced and explored with the aid of the computer, such as solving the Bessel equations (see Chapter 2, Exercise 9). The student may trust the system to handle the computations that they themselves are not capable of performing manually, and may rely on more fundamental understandings of the mathematical elements in question (differential equations in the case of the Bessel equations) to verify the results of these computations. A common technique with differential equations is to simply substitute the solution into the differential equation itself. Multiple avenues of such verification may be employed, reinforcing the student's understanding of the core concepts. In this way advanced mathematics may be introduced earlier than would have been previously possible.

Nevertheless, there was a thin line to be trod. Examples and problems could quickly become intractable for students, even with the aid of the system. Conversely, if problems were not sufficiently difficult or time consuming to solve by hand, then the student may not be sufficiently motivated to learn the new computational techniques. In one particularly memorable example, a student answered an exam question regarding the largest element of a Fibonacci-like sequence less than 1,000,000 (see p204 in the text) by essentially manually computing approximately twenty nine additions, one input block at a time.

Other considerations for the text were: 1) selection and ordering of material in such a way as to be accessible to the student and self-contained, and 2) adherence to a narrative, both of which needed to take practical considerations into account. For example, it was originally envisaged that the Linear Algebra chapter would precede the Calculus chapter, as the concepts are somewhat simpler, or at least more directly applicable to CAS computation. However, our students see calculus before they see linear algebra and so are more familiar with it. Furthermore, many students took the Linear Algebra course simultaneously with the MATH2600 course, and so delaying the Linear Algebra portion of the course (and hence the text) was more pragmatic.

Some students who are familiar with programming have wondered why they would use *Maple* instead of their language of choice. This is a reasonable question; much of the material covered in the first chapter can be performed almost as easily, and with quicker execution times, in a compiled language (such as C, C++, or Java). The algebraic nature of *Maple*'s language combined with manipulation of sets and lists provide accessibility to students with little or no programming background, largely bypassing many of the technicalities and confusions common when learning to program. To the student already comfortable or competent with programming, such features may seem less appealing.

A more significant capability of *Maple* to distinguish it from traditional programming languages[1] is that of symbolic computation. For example, the **sum** and **product** functions from Chapter 1, as well as the recurrence recursion solver **rsolve**. However, it is in Chapter 2 that symbolic computation is most readily seen, with: limit calculations (**limit**), differentiation and integration (**int**, **diff**, **D**), integral manipulation via the IntegrationTools package, and differential equation solving (**dsolve**). Students both strong and weak in programming may benefit from symbolic computation.

Herein, we discuss notable considerations of the decision making process, as well as aspects of the content of the texts or the course material that may not be immediately apparent at first glance.

1. Number Theory

The intent of this chapter is to introduce students to the new *Maple* environment with either familiar or readily accessible mathematics. As such, this chapter deals with *elementary* number theory, as well as some concepts from first year discrete mathematics. Mathematics involving integers, primes, or other discrete mathematical objects (such as recurrence relations) should be accessible to the students mathematically, allowing them to concentrate more on how *Maple* works, and how it can help them better see mathematical concepts and ideas.

A related secondary goal is to introduce new (but still easily accessible) concepts to the students which the computer could help to illustrate. It is with this secondary goal in mind that continued fractions are introduced and explored (§1.3.2), as well as the Sieve of Eratosthenes (§1.3.4). We discuss these examples specifically in detail below.

1.1. §1.1 "Introduction to *Maple*". Although the overriding philosophy of the texts is to use mathematics to motivate the computer code (and not the other way around), before this can be done the computer code needs to be introduced. As such, the entire first subsection (§1.1) deals with introducing *Maple* for its own sake, without any motivating mathematical problems. The chapter establishes what the author considers to be elementary "building blocks" of *Maple* (and, to some extent, programming in general):

[1]Distinguishing in terms of capability readily present in the core language or standard libraries.

expressions, variables, functions, sequences, lists, and sets. It is with combination of these basic tools that the student will do everything else.

It should be noted here that some of the elementary concepts, such as variables and functions, will be immediately familiar to any student or reader with experience in any programming language. Such a background is not necessary and is not assumed by the authors. In some ways, experience in programming can be a hindrance as much as a help. There are problems whose implementation in *Maple* is much more elegant, and even computationally faster if the user thinks like a mathematician and makes use of *Maple*'s inbuilt mathematical structure; for example, using sets to implement the Sieve of Eratosthenes (see Section 1.3 in this document).

In a similar vein, we note that *Maple* variables offer greater utility than variables as used in imperative programming languages. Algebraically speaking, a variable is simply an unknown, whereas a variable in an imperative programming language is simply storage. Inasmuch as *Maple* is a computer algebra system it provides both capabilities simultaneously; *Maple* variables may have values assigned to them, and at the same time any unassigned variable is treated as an algebraic unknown. Additionally, a *Maple* variable may store any valid *Maple* expression, further distinguishing them from the more usual programming variables whose assigned values are often required to be of a pre-determined type. In practice, these differences do not cause any problems either for students familiar with programming, or those that aren't. It is nonetheless a complication worth considering.

There was some question as to whether to introduce sums and products in this section or in the next section. On the one hand, sums and products are quite elementary mathematical objects from a first year calculus point of view. On the other, from the *Maple* point of view sums and products are put together using the building blocks established earlier in the section, and so are arguably not basic elements in their own right. Ultimately, sums and products introduce notions of inert and active forms of mathematical objects, which appear in several other commands later on when dealing with Calculus. As such, they were kept in this section.

One of the insights from teaching the course is that much of the learning needs to be "hands-on". This is particularly true of the early introduction to the system, and thus the goal of the very first subsection. Many stumbling points were remarkably difficult to explain in prose and to demonstrate with static examples. Similarly, there were also some technical points of *Maple* which were useful to know, but were really too obscure or tangential to include in the main body of text. Such points were deemed better demonstrated and introduced through exercises, rather than through prose.

A particularly illustrative example is Exercise 3(b), dealing with a peculiarity of the % operator, which represents the value of the most recently completed computation. Most, of the time this will be the result of the computation at the previous input block. However, due to the dynamic nature of the *Maple* system, this could just as easily be a computation sitting much earlier, or later in the worksheet file, which a user might have executed out of sequence for any number of reasons. The experience of the author, both personally as well as through teaching students, is that it is very easy to forget the out-of-sequence nature of such an action, causing confusion when a command involving the % operator gives unexpected results. Unfortunately, explaining this phenomena in prose proved to be difficult, whereas having the students discover it themselves in a somewhat directed exercise proved to be a much better approach. A thorough explanation of the intent of each exercise can be found in Appendix A.

An important lesson discovered by teaching the course is that showing examples of when and how commands can fail can sometimes be as important as showing when and how commands work. This principle can be seen by its absence at the end of §1.1.4, on page 9. A claim is made that the **union** and **intersection** keywords work as expected with sets, but do not work at all with lists. The text shows an elementary union and intersection of sets, but neglects to demonstrate the failure of the same commands with lists. The oversight of this example before publication is unfortunate, however an example of such failure conditions being demonstrated can be seen on page 10 (§1.1.6) demonstrating the failure of the **Reverse** function in the absence of the appropriate package. Such examples are used frequently when teaching the MATH2600 course.

Note that this concept need not apply only to a function or command failing outright. In many cases throughout the text examples of a function or command providing unexpected or confusing answers can be seen, and are often used to explore the mathematics of the example in question. No such examples are found in §1.1.1, however, due to the introductory nature of the subsection. Examples from following sections of the text will be highlighted and discussed below, as appropriate.

1.2. §1.2 "Putting It Together". Once a familiarity with the *Maple* environment and the basic "building blocks" of the *Maple* language are established, we begin to put them together and show how *Maple* can be used to tackle more complicated mathematics. The goal of this section is, as with the previous section, to introduce *Maple* concepts and techniques. Specifically, the creation of functions, loops (**for** statements), decisions (**if** statements), nested loops and decisions, recursive functions, and timing.

The more complicated nature of the techniques being shown lent themselves to demonstration with less contrived mathematics, as compared to those used in §1.1. The mathematical concepts were chosen specifically to demonstrate the *Maple* concepts and techniques in question, in contrast to those used in the remainder of the text. However, even with these examples we begin to see opportunities for learning new mathematics, or for demonstrating related concepts which come up in the course of computation.

The computation of partial sums, for example, proved useful for introducing the creation of functions, as well as demonstrating the great utility of a function. Once the partial sum function is created, the student can easily and quickly substitute values into the partial sum formula to compute partial sums. The student may even use the function as a variable storing the partial sum formula, as is seen in the first example on page 14.

The choice of partial sums, however, was primarily motivated by its utility in demonstrating **for** loops. Numerically computing many partial sums, sequentially, and displaying their values can give clues regarding its convergence (or divergence). Such a technique is certainly not diagnostic, however. Nonetheless, it is a useful technique for introducing **for** loops using mathematics from first-year calculus courses. This in turn allows us to introduce or recall p-series, and demonstrate the difference in computation efficiency between $\sum_{k=1}^{\infty} k^{-2}$ and $\sum_{k=1}^{\infty} k^{-\pi}$.

Additionally, the partial sum example allows the demonstration of a bug in *Maple*'s infinite product evaluation. When *Maple* computes $\prod_{k=3}^{\infty} \cos(\frac{\pi}{k})$ it gives an answer of 0, but the correct answer is actually an irrational number closer to 0.1149 (§1.2.1 pp18–21). The demonstration of this bug leads to the introduction of a technique to convert between infinite products and infinite sums, which many students have not seen prior to taking this course. More importantly, perhaps, the example demonstrates that *Maple* can make mistakes, and that one should keep their wits about them at all times. It also demonstrates using multiple angles of inquiry to check results.

A satisfying example to illustrate the use of *Maple*'s **if** keyword was a little difficult to find. After some trial and error, the example of divisors was decided upon. The example of divisors is useful because it incorporates both loops and decisions, and demonstrates how the two interact. The example also demonstrates the properties of divisors and how they relate to the square root of a number, leading to a brief discussion of computational efficiency. The reader sees the interaction between pen-and-paper mathematics with *Maple* and how each can affect the other.

The discussion of divisors introduces a technique that is used several times later in the text. The technique is to construct mathematics from first (or, at least, earlier) principles before introducing a *Maple* command that will perform the calculation directly. This technique is explicitly pointed out to the reader on page 24 of the text.

Furthermore, discussing divisors allows the text to move on to introducing and talking about perfect, abundant, and deficient numbers. Identifying such numbers and classifying numbers in this way neatly provides an example with which to show nesting of decisions. In addition, the computations for these types of numbers are complicated enough that they are not easily implemented as a function using the previously introduced arrow notation. This difficulty is exploited to segue to the subsection on introduction procedures (§1.2.4) while maintaining the narrative of the mathematical topic of perfect numbers.

The topic of recursive functions was added after the first iteration of teaching the course. During that time, an iterative Fibonacci number calculator was written as part of the introduction of procedures. The pseudocode for this procedure can be seen in Algorithm 1. Although the code should be easy to understand for anybody familiar with programming, in practice students tended to struggle to comprehend it.

The recursive procedures in §1.2.6 for calculating Fibonacci numbers were found to be more easily understood by the students. Their advantages are that they are simpler, easier to read and understand, and look more like the usual mathematical definition of the Fibonacci numbers. The tradeoff for these benefits is either computational efficiency, which leads to the discussion and measurement of execution times in §1.2.7, and the subsequent introduction of the **remember** option for procedures.

Algorithm 1: Iterative implementation of the Fibonacci number calculator.

```
Input: $N \in \mathbb{N}$
begin
    $f_{n-1} \leftarrow 1$
    $f_n \leftarrow 1$
    for $k \leftarrow 3$ to $N$ do
        $temp \leftarrow f_n + f_{n-1}$
        $f_{n-1} \leftarrow f_n$
        $f_n \leftarrow temp$
    end
    return $f_n$
end
```

Recursive functions combined with the **remember** option introduces another limitation which is difficult to explain at a level accessible to the intended reader of the texts. In order to avoid unterminated loops appearing as part of an infinite backwards recursion (from, say, neglecting to specify an initial condition), *Maple* limits the number of recursive calls which can happen before it cancels the computation and issues an error. This is the limitation referred to at the bottom of page 36 in §1.2.7. It proved impossible to find an argument to the f function that took approximately a second to compute, as computing more than the 2000th Fibonacci number resulted in too many recursions.[2]

Trial-and-error is quite difficult with this function as well, because every execution causes function values to be remembered, which affects the computation times of later computations (e.g., the time to compute $f(4000)$ after having only computed $f(2000)$, as seen on page 40). Furthermore, the remembered values also allow for larger arguments to be computed before the recursion limit is reached. For instance, the aforementioned computation of $f(4000)$ would have been impossible without the prior computation of $f(2000)$. This forces constant re-setting of the function, in order to properly ascertain execution times, and where the number of recursions exceeded *Maple*'s limit. Such trial and error is easy to show and explain in person, but proved sufficiently difficult in prose as to be avoided in the main text. Instead, Exercise 19 was constructed to demonstrate the problem.

1.3. §1.3 "Enough Code Already. Show Me Some Math!". With this section, the text finally dispenses with introducing *Maple* concepts for their own sake and begins exploring mathematical concepts. There are a mix of topics, some of which are familiar from first year discrete mathematics (induction and recurrence relations), and others of which introduce new, but accessible, concepts (continued fractions and the Sieve of Eratosthenes). For the most part the author feels the subsections speak for themselves, and consequently we discuss only technical and teaching points.

[2]Specifically, when executed immediately after a fresh definition of the function, $f(2934)$ may be computed, whereas $f(2935)$ will cause a "too many levels of recursion" error. This is on *Maple* 15 on the author's home computer. *Maple* 16 on the author's office computer appears to allow more recursions, exhibiting the same behaviour for $f(11000)$ and $f(12000)$ respectively.

Continued fractions, while quite simple, can be tedious and time consuming to compute by hand. Using *Maple* allows the construction of a continued fraction to be explored using basic mathematical principles of what a continued fraction is, combined with the loop structure of the *Maple* language which was introduced earlier in the chapter. The choice of a finite decimal to begin with (0.123456789) permits an easy check of the resulting continued fraction by simply typing the fraction into *Maple* and seeing that it is reduced to 123456789/1000000000. The "pretty-printing" nature of *Maple*'s worksheets means that the students can also see the continued fraction printed in all its glory. Regrettably, this simple check did not make it into the text, although it has become a standard part of the lecture when teaching the subject face-to-face.

Such checking, based on early principles, has become a staple of the face-to-face teaching and testing of the course. Students are routinely asked questions (usually recurrence relations or differential equations) which they are almost certainly incapable of solving by hand, but for which *Maple*'s inbuilt functions can provide a solution. It is expected (and, during lectures, demonstrated) that the student can verify the solution by simply substituting it back into the original formula and checking the truth of the result.

The Sieve of Eratosthenes was chosen partly because of its simple elegance and partly because the implementation demonstrated in the text contains a nifty computational idea: an array of boolean values wherein the index to the array as information. If the boolean value at address n of the array is *true*, then the number n is prime (and, conversely, if the value is *false*, then n is not prime). The implementation is very much the sort that a C or C++ programmer would use, and in practice some students struggled to understand its technicalities. Note that if a list is used instead of an Array, then *Maple* issues an error about assigning values to long lists and requires the use of an Array; regrettably this was not mentioned in the prose.

We note that the need for an alternative impelementation of the Sieve of Eratosthenes for the *Mathematica* text (see Section 1) led to finding a simpler implementation (and a superior one, in the opinion of the author) that has since been taught in the face-to-face course. The new implementation uses set manipulation, removing multiples of a newly found prime via the set **minus** operation. Some technicalities can still crop up with this implementation, such as keeping track of what the "next" uncrossed out number is, but in practice the set-based implementation is shorter, easier to read, easier to understand, and, surprisingly, faster in execution. Perhaps more importantly, it allows the student to think more like a mathematician, and less like a programmer, which in practice is beneficial for many students.

This demonstrates a certain dichotomy of thinking with regards to using *Maple*: mathematical vs programatic. Inasmuch as the background of the author before studying mathematics included work as a computer programmer, many of the *Maple* examples used for the first iteration of course notes tended to use techniques similar to those employed in programming languages such as C, C++, or Java. Such techniques proved difficult for less technically capable students. We have found that a strong advantage of *Maple* (or *Mathematica*) as a system for performing mathematics is that its inherent mathematical nature allows mathematically-minded people who might otherwise struggle with the deep technicalities of programming to nonetheless use a computer as a tool of mathematics. As such, examples in the texts have progressively moved away from such programatic thinking toward a more mathematical thinking, and we note that this trend has continued in lectures after the publication of the texts.

2. Calculus

For the most part, the Calculus chapter of the text looks at calculus topics at a level which ought to be familiar to the student from first year. We do so for two reasons. Firstly, some students taking the course have only marginally passed the assumed knowledge from first-year calculus and revising this material with the added benefit of having the software to help with computations should reinforce comprehension. Secondly, students who are strong in the assumed knowledge stand to benefit from new perspectives on the familiar material made possible by the use of *Maple* for demonstration and exploration. In both cases, the aid of the computer allows the student to approach more difficult or complicated problems than they could previously have done by hand.

The chapter is divided into three sections. The first covers calculus from first principles and introduces *Maple* commands for dealing with limits, integrals, derivatives, and the visualisation functions whose domain and ranges are real. The second covers applications of univariate calculus. The third covers elementary multivariate calculus, and is new material for some students who take the course. In practice, the entirety of the chapter is significantly more than can be covered in the three or four lectures devoted to teaching the Calculus portion of the course.

We note the following items of particular interest relating to the content of the chapter and its development. It is assumed that the text otherwise speaks for itself.

Visualisation is an incredibly useful tool in calculus. As such, §2.1.1 and §2.1.2 are devoted solely to introducing *Maple*'s **plot** function and its vagaries. Note that §2.1.1 is in the same style as §1.1 and §1.2 in that the goal is to introduce *Maple* code, however §2.1.2 introduces its concepts through an inequality problem which would likely be inaccessible to the target audience without the aid of visualisation.

When introducing differentiation from first principles, the text constructs a procedure that prints the limit in inert form, as well as the value of the limit. Using this function to compute the derivative of $\sin x$ at $x = \pi/2$ outputs the limit $(\cos h - 1)/h$ which has a $0/0$ form, yet converges to 0 as $h \to 0$. In lectures, a similar but extended approach is sometimes taken, where the derivative of sin is explored through first principles, resulting in the limit

$$\sin(x) \lim_{h\to 0} \left(\frac{\cos(h) - 1}{h} \right) + \cos(x) \lim_{h\to 0} \left(\frac{\sin(h)}{h} \right)$$

thus reducing the derivative of sin to the limit of $\sin(h)/h$ as $h \to 0$ (since $(\cos(h) - 1)/h$ can be manipulated into an expression involving $\sin(h)/h$). The proof of the value of the limit hinges on the fact that $\cos(\theta) \le \sin(\theta)/\theta \le 1$ for $0 \le \theta \le \pi/2$, which can be nicely visualised using techniques similar to those in §2.1.2. Visualising this inequality also provides a nice demonstration of the squeeze principle.

The “flower plot” on page 102 (§2.2.5) lends itself very nicely to an animated plot, to better explain the cyclic nature of the function being plotted. Regrettably, such dynamic visualisation does not work particularly well when printed in a book. The plot is shown in lectures, when the extra plotting material is covered. Additionally, animations of Taylor series of increasing degree are also shown (along with the original function) to demonstrate Taylor approximations, which is a topic not covered in the texts at all.

The visualisations of surfaces and volumes of rotation produced on pages 109 and 110 (§2.3.2) were directly motivated by a mistake made by the author during the first draft of the notes. After converting the function $x = z^2$ to $z = \sqrt{x}$, the author lost track of which area was being rotated to create the volume, and thus inadvertently computed the area between the curve and the z-axis (instead of the area between the curve and the line $z = 2$). To compound matters, the error did not show up when checking the value computed against the “shells” method of computing the volume because of a curious property of the paraboloid, where the volume obtained by rotating the area between the curve and the z-axis around the z-axis happens to be exactly the same as the volume obtained by rotating the area between the curve and the x-axis around the z-axis. This led directly to the visualisation on pages 109 and 110, as well as to Exercise 11(a).

3. Linear Algebra

The Linear Algebra chapter follows a traditional treatment on the topic. We assume no prior exposure to vector spaces, however, we do assume that the student has seen introductory vectors (usually $\mathbb{R}^n$) along with elementary vector arithmetic and the dot product. It is also assumed that the students have seen matrices before and know how to multiply them (and, consequently, how to multiply a vector and a matrix together). These topics are all covered in first year calculus courses at the University of Newcastle, without the framework of vector spaces.

The first section (§3.1) reviews this assumed material, beginning by showing how common vector and matrix arithmetic procedures are performed in *Maple*, and then following up the usual treatment with the added aid of visualisation. A student’s understanding of planes, in particular, benefits greatly from visualisation, as there are many configurations of simultaneous planes that can be difficult for beginning students to conceive of initially. For example, the case where three or more planes intersect at only a single point, or the case where two or more planes fail to have any points in common. Only one such example is given in the text, although Exercise 3 (p180) demonstrates some other configurations.

An unusual aspect of the first-year linear algebra review in is found in §3.1.3, where elementary matrices are introduced and used to prove that row reduction can be used to find matrix inverses. The technique of finding the inverse of a square matrix M by row-reducing an augmented matrix $[M|I]$ (where I is the identity matrix) is taught in first year at the University of Newcastle, however a proof of the correctness of the technique (or explanation as to why it works) is never taught in fist year and is rarely taught in the second year Linear Algebra course.[3] The proof is relatively simple, and the use of *Maple* significantly helps in demonstrating the techniques involved which are onerous and error prone if performed by hand due to of the large number of matrix multiplications involved.

The remainder of the chapter (§3.2 and §3.3) follows a standard introduction to Linear Algebra and covers: the introduction of vector spaces, linear combinations, linear dependence, basis and dimension, linear transformations, matrix representation of linear transformations, eigenvectors and eigenvalues, and diagonalisation. The choice of topics and the breakdown of the two subsections were due to the time constraints of lecturing (approximately one lecture per chapter), although in practice the material presented tends to require more time. Since the goal of the course as a whole is to introduce students to computer assisted mathematics, the lectures on this topic finish at whatever topic is arrived at after three for four weeks of lecturing, with little concern as to whether some topics were missed.

Of particular note is the tendency in both the text and the lectures to show examples of the linear algebra concepts in vector spaces other than $\mathbb{R}^n$. Polynomial spaces ($P_n(\mathbb{R})$ or $P_n(\mathbb{C})$) are predominantly used, and matrix spaces are left as challenging exercises. All of linear combinations, dependence, basis, dimension and even linear transformations and matrix representation of linear transformations are demonstrated on polynomial spaces. The demonstration is little more than exploiting the fact that every finite-dimensional vector space over a field $\mathbb{F}$ is isomorphic to $\mathbb{F}^n$ for some n, which is the essence of performing linear algebra in *Maple*. The demonstration is aided by the fact that most of these concepts come down to solving a linear system, and *Maple* allows for quick and easy solution of linear systems, as well as quick and easy transformation between the vectors in question and their isomorphic images in $\mathbb{F}^n$.

The notion of isomorphism is not directly addressed in the text. Instead, the isomorphism between $P_n(\mathbb{F})$ and $\mathbb{F}^n$ is initially described as a "correspondence" (p149) with the isomorphism itself described explicitly in terms of the coefficients of the polynomials and the components of the vectors. After the concept of vector bases is introduced, this correspondence is explained as one between basis vectors, allowing for an analogous procedure to be performed between any finite-dimensional vector space and $\mathbb{R}^n$ for appropriate n. Nonetheless, the omission of directly talking about isomorphisms between spaces is an oversight which ought to be corrected should a second edition ever be produced.

4. Visualisation and Geometry

The Visualisation and Geometry chapter forms a postscript to the text. At the time of writing the initial course notes, members of the CARMA research group at the University of Newcastle, including Professor Jon Borwein, were using the *Cinderella* interactive geometry package for some of their research. Inasmuch as *Cinderella* is designed for more elementary geometry, its utility for research in optimisation is remarkable. Consequently, it was used when teaching the course, both to introduce the software and to highlight the benefits of interactive computation.

[3]At least, this was the case at the time of writing the first round of course notes.

Regrettably, the interactive nature of the software lent itself more to demonstration and direct experience than it did to explanation in prose. As such, the Visualisation and Geometry chapter was co-opted to introduce some plotting and visualisation tools that didn't come up directly in the mathematical exploration of the text but which were nonetheless useful. Additionally, we demonstrated some geometric constructions using *Maple*'s inbuilt geometry package which, despite not being interactive, was much better suited to explanation and demonstration in prose.

CHAPTER 3

Mathematica

When the project was extended to include *Mathematica* a new set of issues surfaced. It was originally envisaged that the process of converting the *Maple* text to *Mathematica* would—more or less—be a simple case of applying the techniques from the original text in *Mathematica*, replacing the *Maple* examples in the text with the requisite *Mathematica* code, and replacing occurrences of the word "Maple" with "Mathematica". It was understood that some of the explanatory paragraphs would need to be modified in some, presumably small, ways.

It turned out, however, that the differences between the two systems were great enough that many otherwise apparently basic techniques needed remarkably different approaches, and consequently different explanations. Although the same structure and material was kept, large amounts of the first chapter needed to be re-evaluated and re-written, as well as smaller—but still significant—parts of later chapters.

We discuss those issues here, chapter by chapter, and note the requisite changes made to address them. The reader may assume that the desires and motivations described for the *Maple* text remain. Note that much of the work was spent in ascertaining how to perform operations in *Mathematica* and then in re-crafting the explanation required in the text. As such, while the content is not radically different, much time was spent in re-crafting it.

An immediate difference between the two systems is in their "pretty-printing". Recall that the decision to use the *Maple* so-called "2D input" mode was (in addition to it being the default on *Maple* version 10 and onward) in order to have the maths on the computer screen look as similar as possible to mathematics as it is written on paper. This is more difficult with *Mathematica*; its output is pretty-printed, but the commands the user inputs tend to be purely textual. It is possible to have the inputs pretty-printed, however, to do so is tedious. In contrast to *Maple*'s method of 2D input, *Mathematica* requires the use of `CTRL-6`, `CTRL--`[1] to produce typeset superscripts and subscripts, and the use of `CTRL-/` to produce typeset fractions. Regrettably, the author has only become aware of this facility after the publication of the text.

Other characters, such as union (∪), intersection (∩), greek letters, and others, require escape sequences to be typed in by first pressing the `ESC` character, followed by a sequence of characters (usually an abbreviated form of the name of the symbol to be input, or its mathematical meaning), followed by the `ESC` key again. For example, the union symbol is achieved by typing `ESC`, `u`, `n`, `ESC`, and the λ greek letter is achieved by typing `ESC`, `l`, `a`,

[1]That is, the control and hyphen (or dash) keys pressed simultaneously.

`m`, `b`, `d`, `a`, `ESC`. Alternatively, the backslash notation may be used for similar effect, such as **`\[Theta]`**, as shown in §4.1.3 (similarly, λ can be achieved with **`\[Lambda]`**).

It was decided that these pretty-printing options were sufficiently convoluted to explain in prose to be worth avoiding. Instead, the input was kept as text input. This is perhaps most noticeable in the Calculus chapter, which contains many polynomials and rational functions.

1. Number Theory

The Number Theory chapter required the most effort to re-work into appropriate *Mathematica* code and related explanation. This should not be surprising, given that this is the chapter in which the most time is spent the most time introducing the system for the first time.

The first point of difference is in the way *Mathematica* handles lists and sets. In *Maple* the system distinguishes sequences, lists, and sets as different data types. In *Mathematica* the system has only the lists data type, however lists may be treated as sets using appropriate functions (**`Union`**, **`Intersection`**, etc). This proved to be a little jarring when learning *Mathematica* while being accustomed to *Maple*; this is likely lessened when learning *Mathematica* as one's first mathematical package. Additionally, the use of the **`Complement`** function to perform what is essentially a set minus operation is a somewhat unexpected choice, as is discussed on page 10.

Note that *Mathematica* does not have alternate "inert" and "active" forms for its sums and products (or anything else for that matter) as *Maple* does. One may achieve a pretty-printed unevaluated mathematical expression using a combination of the **`HoldForm`** and **`TraditionalForm`** functions. The **`Hold`** function is used to prevent *Mathematica* from evaluating an expression, and the **`ReleaseHold`** function will undo a hold on an expression (i.e., **`ReleaseHold`** will cause a held expression to evaluate). However, an expression **`<expr>`** when applied as an argument to the **`Hold`** function will output as "Hold[<expr>]", whereas the output of the same expression as an argument to the **`HoldForm`** function is simply the unevaluated expression. Pretty-printing is only applied to the output in some cases, but the effect can be achieved with the **`TraditionalForm`** function when needed. These functions, while useful, were deemed to be too troublesome to use routinely for the purposes of the material being covered in the text. The functions are mentioned in passing in a couple of locations (pages 56 and 85) and were left to the interested reader to pursue further.

The next point of difference is that *Mathematica* allows for prefix, infix, and postfix notations for function calls (see §1.1.6). These alternate notations could have been omitted entirely and the text likely would not have suffered, especially when we consider that *Maple* uses none of these functional notations. However, we found that judicious use of these notations helped to make code more readable, and so we introduced them.

Function creation in *Mathematica* uses pattern matching, which is more generally powerful than the creation of functions in the sense explored in §1.2.1. The concept of pattern matching is expanded upon in §1.2.2 and §1.2.3, and briefly in §1.2.7. It should be noted that the text does not use the full extent of *Mathematica*'s capability for pattern matching.

Note also that *Mathematica* includes what it refers to as "pure functions", defined using either the `&` operator or the `Function` function. Pure functions are conceptually similar to *Maple*'s arrow function notation (as introduced in §1.2.1 in the *Maple* text) in that they may be used to specify a function in-line without having to assign it to a variable name first. Such a technique is used very infrequently in the *Maple* text (see §2.1.4, §2.1.5, and §3.1.1). In the author's opinion, pure functions were sufficiently confusing to avoid in favour of the pattern matching technique (which also opened the way to further explore pattern matching in later subsections). This led directly to the changes in §1.2.4 from the discussion on *Maple* procedures in the original text to a more pattern-focussed discussion in the *Mathematica* text.

A discrepancy in the way *Maple* and *Mathematica* compute certain p-series forced the removal of an example from the text. *Mathematica*'s calculation of the partial sums $\sum_{k=1}^{N} 1/k^2$ evaluate only to rational numbers (see §1.2.1). In the *Maple* text, the 10,000th partial sum is computed as $-\Psi(1, 10001) + \pi^2/6$, however *Mathematica* will compute the same partial sum a fraction whose numerator and denominator each contain 8693 digits.[2] Furthermore, *Mathematica* doesn't have as nice a formula for the Nth partial sum as *Maple* does (we see this in §1.2.2). The discrepancy essentially invalidated the exploration of $\sum_{k=1}^{\infty} 1/k^{\pi}$ in the *Mathematica* text because the partial sums of the two p-series take approximately the same amount of time to compute in *Mathematica.* This is not the case in *Maple*; in fact the difference in computation times is precisely the reason for exploring the second sum in the *Maple* text. Consequently the $\sum k^{-\pi}$ exploration was omitted from the *Mathematica* text.

In §1.2.1, the exploration of the computation of $\prod_{k=3}^{\infty} \cos(\frac{\pi}{k})$ was removed because *Mathematica* did not produce the same erroneous value of 0 that *Maple* computed. In the *Maple* text, this mistake serves three purposes: 1) as a reminder that the computer is not infallible, 2) as a demonstration of multiple avenues of exploration and checking, and 3) as a convenient excuse to introduce the idea of converting between sums and products using logarithms and exponentials. Regrettably, attempts to frame these points in the *Mathematica* text felt contrived without the motivating computational mistake, and so the example was dropped.

Note also that *Mathematica*'s `For` loops are significantly different to *Maple*'s. The *Mathematica* `For` loop would be familiar to a C (or C++) programmer and consists of initialisation, test, and incrementation code appearing before the code block that is to be executed by the loop. It was decided that this was probably too technical to introduce to readers at the level expected by the text. Fortunately, *Mathematica*'s `Do` and `While` loops were much more straightforward and used the familiar iterator syntax introduced in §1.1.4 and §1.1.5. Furthermore, these loops were conceptually similar to those seen in *Maple*. Consequently, the `Do` and `While` commands were used instead of the `For` command because they required less rewriting and re-explaining. We note that in all cases the loops

[2]This computation was not included in the text due to the fraction being too large to print.

(and decisions as well) in *Mathematica* are all functions, as opposed to *Maple* where loops and decisions are part of the syntax of the language.

There is a bug that arises with recursive functions in the event that `$RecursionLimit` is set too high and this caused some problems with the exploration of Fibonacci number computation times in §1.2.7. We were able to cover sufficient material in the section that we did not need to change the content at all, and simply reported the bug (see page 48). We note here that the bug is still apparently present in *Mathematica* version 9.0.1.0.[3]

The final significant point of difference in Chapter 1 was to do with the Sieve of Eratosthenes. The implementation used in the *Maple* text was exceptionally difficult to achieve in *Mathematica*. Lists cannot easily be modified in place like a *Maple* array can. Furthermore, despite many of *Mathematica*'s idiosyncrasies being similar to those of a language such as C or C++, it apparently lacks anything like a C array (to which a *Maple* Array behaves similarly). As such a different implementation was required. During the early stages of converting the *Maple* code to *Mathematica* code, a visiting undergraduate research fellow from the California Institute of Technology, Joshua Borwein, experimented with a recursive implementation that also used sets. Ultimately this implementation proved to be too cumbersome and confusing for the target audience, but the idea of leveraging sets (instead of lists) led the author to the implementation that was ultimately adopted. The set-based implementation also proved to be effective in *Maple*, as discussed in Section 1.3 above.

2. Calculus

With the introduction to *Mathematica* out of the way, fewer changes were required to be made to the content of the text. For the most part, the exercises were able to stay the same and it only remained to find analogous *Mathematica* code to perform the steps performed in *Maple* in the original text. We note the following additional points of interest.

Mathematica's plotting of trigonometric functions was better behaved than *Maple*'s. Specifically, the plot of the tan function in §2.1.1 (p79) did not produce the same erroneous plot that appears in the *Maple* text in the same section (p69). We were able to replicate the incorrect plot in *Mathematica* with the use of the `PlotRange->Full` option to the `Plot` function. Doing so introduces the `PlotRange` option and also allows the discussion of the vertical axis scale. We note, however, that it is a little less satisfying to start with a working function and have to break it in order to introduce discussion, than it is to start with a broken function and discuss how to fix it. Interestingly enough Version 16 and later of *Maple* no longer produce the erroneous plot (and, in general, these versions are better at choosing vertical ranges automatically).

The lack of a convenient "inert" form for mathematical objects (as discussed above) caused some problems. In §2.1.3 we briefly show the reader how to print the limit in traditional mathematical notation and explain that we will avoid the technique. A more significant problem lies in the treatment of symbolic manipulation of integrals, as discussed

[3]The *Mathematica* version used during the writing of the text was version 8.

in §2.2.3 of the *Maple* text. We were forced to abandon the entire sub-section for the *Mathematica* text.

We also note that *Mathematica* lacks a command analogous to *Maple*'s **identify** function. As such, the discussion at the end of §2.2.2 dealing with the erroneous identification of 16.37297620 (the computed value of $\int_0^\infty x^2/\sqrt{e^x-1}\,dx$) as $(9/2)2^{3/5}\sqrt{3}\zeta(5)^9$ was, sadly, reduced to a paragraph describing in prose the results of the exploration conducted in the *Maple* text.

We also note that the integral $\int_0^\infty x^2/\sqrt{e^x-1}\,dx$ was able to be evaluated directly by *Mathematica*, whereas *Maple* (as of version 16) remains unable to evaluate it without the substitution given in the text. The corresponding substitution feels weak in the *Mathematica* text, as it was not needed to evaluate the integral. We note that this was a common theme with *Mathematica*; it tended not to make mistakes similar to the ones *Maple* made that we were able to exploit for teaching purposes. *Mathematica*'s quirks and shortcomings tended to be very technical, to be hidden from the user, or to be esoteric mathematically.

3. Linear Algebra

The Linear Algebra chapter required the fewest changes to be made. The only significant point of divergence from the *Maple* text is the fact that *Mathematica* does not have inbuilt matrix or vector types, instead it simply uses lists in particular configurations. A result of this is that it does not automatically print matrices and vectors in a mathematical (read: pretty-printed) manner. Pretty-printing is achieved via the `MatrixForm` function for both matrices and vectors, however, once a vector or matrix is pretty-printed in this way, *Mathematica* no longer recognises it as a list (and consequently as a matrix or vector). As such, pains needed to be taken to make sure that pretty-printing intermediate results didn't interfere with computations. In practice this wasn't difficult.

This technicality did, however, allow a reasonably easy way to output an unevaluated matrix multiplication. Such a technique is useful during lectures to show students, on screen, the matrices to be multiplied, and to aid in checking simpler results manually. We see an example of this in §3.1.1 (pp141–142). With *Maple*, the author usually uses a sequence to achieve a similar result.

4. Visualisation and Geometry

The Visualisation and Geometry postscript chapter required significant rewriting. It was decided that mention should be given to *Mathematica*'s demonstrations library and also to *Mathematica*'s interactive capabilities. However, due to the limitations of explaining and demonstrating interactive elements in static prose, this amounted to a paragraph's mention each, containing information for where the interested reader should go to further pursue the topics. Additionally, a brief demonstration of animation was added. The remainder of §4.1 was essentially unchanged.

A significant difference stems from the fact that *Mathematica* lacks any sort of geometric package similar to that used in §4.2 of the *Maple* text. Despite the above difficulties, It was decided that the interactive geometry package *Cinderella* should be introduced and used. This was achieved by giving the interactive nature of the software a description in the introductory paragraphs of §3.2, along with the limiting factor of explanation in prose. The constructions were then described with accompanying images generated in *Cinderella* in place of the *Maple* code that was present in the *Maple* text.

CHAPTER 4

Epilogue

In summary, we have produced a course and texts teaching introductory computer-assisted mathematics and computational mathematics in which the primary focus is on the mathematics with the tool in question (*Maple* or *Mathematica*) acting as more of a means to an end. It is hoped that upon completion of the course or one of the texts, the student or the reader is able to apply principles of learning and exploration to other mathematical concepts and computational tools.

We close by noting that the course continues to be taught and continues to evolve. Students invariably end up showing the author something new; sometimes directly and sometimes indirectly. Furthermore, the author uses these tools in his other research. Both situations lead to learning new techniques or subtleties of the systems, some of which make their way into lectures.

Nonetheless, the topics and techniques published in the texts remain a stable, solid foundation with which to teach the course. This is in no small way due to the iterative nature of teaching the course and writing the material. Currently, the author requires students to read relevant sections in their own time and gives lectures on complementary material. The complementary material is either clarification of students' misunderstandings, greater depth and detail regarding the material in the relevant section, or covers a topic of mathematics related to the material in the relevant section. Student feedback has been uniformly positive.

References

[1] ARATYN, H., AND RASINARIU, C. *A Short Course in Mathematical Methods with Maple.* World Scientific, 2006.

[2] BAILEY, D., BORWEIN, J., CALKIN, N., GIRGENSOHN, R., LUKE, R., AND MOLL, V. *Experimental Mathematics in Action*, 1st ed. AK Peters, Wellesley, MA, 2007.

[3] BORWEIN, J., AND BAILEY, D. *Mathematics by Experiment: Plausible Reasoning in the 21st Century*, 2nd ed. AK Peters, Wellesley, MA, 2008.

[4] BORWEIN, J., BAILEY, D., AND GIRGENSOHN, R. *Experimentation in Mathematics: Computational Paths to Discovery*, 1st ed. AK Peters, Natick, MA, 2004.

[5] BORWEIN, J., AND DEVLIN, K. *The Computer as Crucible: An Introduction to Experimental Mathematics.* AK Peters, Wellesley, MA, 2009.

[6] GANDER, W., AND HŘEBÍČEK, J. *Solving Problems in Scientific Computing Using Maple and MATLAB*, 4th ed. Springer, New York, 2008.

[7] GARVAN, F. *The Maple 5 Primer Rel 4.* Prentice Hall, Englewood Cliffs, NJ, 1997.

[8] GARVAN, F. *The Maple Book.* Chapman and Hall/CRC, Boca Raton, FL, 2001.

[9] GRAY, T. W., AND GLYNN, J. *Exploring mathematics with Mathematica.* Addison-Wesley, 1991.

[10] HECK, A. *Introduction to Maple*, 3rd ed. Springer, New York, 2003.

[11] ROVENSKI, V. Y. *Geometry of Curves and Surfaces with MAPLE.* Springer, New York, 2000.

[12] WAGON, S. *Mathematica in Action*, 2nd ed. Springer, New York, 1999.

APPENDIX A

Exercise Guide

Included here is a Lecturers guide to the exercises for "An Introduction to Modern Mathematical Computing with Maple" and "An Introduction to Modern Mathematical Computing with Mathematica". The guide is current as of the date of printing of this document. An updated guide (if there is one) can be found at `http://carma.newcastle.edu.au/books/mathematicalcomputing/`.

An Introduction to Modern Mathematical Computing with *Maple*

This document is intended to act as a guide to the exercises in the book "An Introduction to Modern Mathematical Computing with *Maple*". It is primarily intended for instructors, lecturers, tutors or the like, and outlines the intent of the exercises, as well as any technical points, stumbling blocks, or other significant point the student or reader might miss. Students should feel free to read the contents of this document, but should be warned that the material is aimed at a higher level than the books are aimed at.

For the duration of this document we will refer to the person attempting the exercises as *the student*, although we acknowledge that not everybody attempting the exercises from the text will necessarily be a student. The book itself, and the exercises therein, was primarily written as a text to accompany a second year university course, and as such the exercises were written with teaching students in mind.

Number Theory.

(1) The goal of this exercise is to give students practice in entering basic expressions into *Maple*, as well as in using the **evalf** function. Note that part (h.) may potentially lead to a discussion as to the precedence of taking powers. That is

$$2^{2^{2^{2^2}}} = 2^{\left(2^{\left(2^{\left(2^2\right)}\right)}\right)} \neq \left(\left(\left(2^2\right)^2\right)^2\right)^2 = 2^{2\cdot 2\cdot 2\cdot 2}$$

Furthermore, *Maple* gives an exact integer answer (albeit with some 19,500 odd digits removed), but rounds rounds the floating point approximation to 10 significant figures (or however many significant figures the student specifies).

(2) This exercise is to show the student how *Maple* (and, specifically, the **evalf** function) handles digit precision of decimal approximations. Note that parts (a.) through (d.) are meant to be demonstrative, and should give the user clues about how to solve parts (f.) through (h.).

Also, note that setting the *Digits* variable changes the precision of the entire worksheet, whereas specifying the precision directly in the **evalf** function only changes the precision for that particular calculation.

(3) This exercise demonstrates two possible stumbling blocks. Firstly (part (a.)) that some variable names are protected in *Maple*. Common names the students will likely end up wanting to use themselves will be I (the imaginary unit) and **D** (the differential operator). This exercise should introduce the idea that some variable names are already taken, and can't be used.

Note that *Maple* provides the capability to remove or grant protection via the **unprotect** and **protect** functions respectively. Doing so should not be necessary in the context of the book, or its exercises, and so these functions are not mentioned at all there.

Secondly, (part (b.)), demonstrates a common misunderstanding with the **%** operator. Specifically, the **%** operator represents the result of the *most recently* performed computation. This exercise demonstrates that the most recently performed computation is not necessarily the computation on the previous line of the workbook.

(4) This exercise is to show the student some of the possible capabilities of the the **convert** function.). Note that parts (a.) through (h.) are meant to be demonstrative, and should give the user clues about how to solve parts (g.) through (k.). The reader should also need to search through *Maple*'s help files to solve some, or all, of parts (g.) through (k.).

Also, note that part (k.) includes a possible stumbling block if the user forgets to explicitly specify that the terms in the denominator are multiplied (either by using , or a space). That is, *Maple* treats $(x-1)(x^2+2)^2$ and $(x-1)\cdot(x^2-2)^2$ differently (the latter has a dot between the parenthesised expressions). The reader should be three expecting fractions with denominators $(x-1)$, (x^2+2), and $(x^2+s)^2$. However, if they forget to explicitly specify multiplication, they instead get a single fraction with denominator $(x(x^2+2)+1)^2$.

The reason for this discrepancy is that *Maple* interprets $(x-1)(x^2+2)$ as the function $(x-1)$ applied to an argument of $x^2+2)$, where the function $(x-1)$ is the function obtained by pointless subtraction of the constant function 1 from the function x. If we call this function g (i.e., $g := x-1$), then $(x-1)(x^2+2) = g(x^2+2)$ and so

$$\begin{aligned}(x-1)(x^2+2)^2 &= g(x^2+2)^2\\ &= (x(x^2+2)-1(x^2+2))^2\\ &= (x(x^2+2)-1)^2\end{aligned}$$

which explains the seemingly strange denominator.

(5) This exercise demonstrates how the **$** operator may be used as a shorthand for the **seq** function. Note that the **$** operator is strictly less powerful than the **seq** function, and the author recommends the use of **seq**. For example, the

sequence produced by $seq\left(\frac{2}{n}, k\,\mathbf{in}\,[2,3,5,7,11,12]\right)$ cannot be produced using the **$** operator (see Exercise 6).

(6) This exercise is to show the student some of the more technical capabilities regarding dummy variables in the **seq** function. Note that parts (a.) through (d.) are meant to be demonstrative, and should give the student clues about how to solve parts (e.) and (f.).

Note that the intent for part (f.) was to have the student to solve the exercise using a list inside the **seq** function (i.e., with $seq\left(\frac{1}{k}, k\,\mathbf{in}[1,2,4,7,11,16]\right)$ or something similar). Some students notice that the denominators exhibit a pattern where first 1 is added, then 2, then 3, and so on. Some careful mathematics should show that the kth term of this sequence is $1 + \frac{n\,(n-1)}{2}$ and so the problem may be solved by $seq\left(\frac{2}{k^2-k+2}, k = 1..6\right)$.

(7) This exercise is to show the student how a sequence, list, or set may be indexed in reverse. Note that the failures are produced because the ranges are backwards. For example, part (g.) is equivalent to asking for $L_{[10..8]}$, which is not a valid range because the left hand side is larger than the right hand side.

(8) This exercise is to demonstrate the **op**, **nops**, and **numboccur** functions, as well as the **Occurrences** function from the ListTools package. The student is expected to look at the help information for the **Occurrences** function themselves, and use that information to work out how to solve parts (m.) through (p.)

Note that *Maple* stores expressions as an expression tree, where the internal vertices are operations, and the children are the arguments to that operation. The **op** command returns the immediate children of the root of the tree, as a sequence, and the **nops** command returns the number of said children. The **numboccur** function is a little different; it returns the number of occurrences of an expression (the second argument) in the entire tree; this is demonstrated in the second part of the exercise, with the nested list. Conversely, the **Occurrences** function counts the occurrences of the expression (its first argument) in the first level of the list (the second argument) only.

In the case of lists, it is expected that students wanting to know how many times a particular element of the list occurs, they will only want to consider the first level of lists, and as such **Occurrences** exhibits the correct behaviour, and **numboccur** does not. However, **op**, **nops**, and **numboccur** are applicable to any expression, where **Occurrences** only operates on lists.

A stumbling block to note is the following. If a list is given to **numboccur** as the sub-expression to count, then **numboccur** will count the occurrences of each individual element of that list, and will return the sum of those counts. For example, $numboccur(E, [s1, s2])$ will count the number of occurrences of *s1* in expression E, as well as the number of occurrences of *s2* in expression E, and will add the two counts together for the final result. To have **numboccur** count the number of occurrences of a sub-expression which is, itself, a list, then that list must be put inside a list. For example, *numboccur*$(E, [[s1, s2]])$ will count the number of occurrences of the list $[s1, s2]$ inside the expression E.

(9) This exercise is a challenge. The student will need to use the help files, and some trial and error to work out how the given **convert** command variant works. Then some insight, and some more trial and error will be needed to work out how to convert the required digits of π into an integer for use with the **convert** function.

Note that this exercise is related to the *further exploration* of normal numbers (§1.5, p65).

(10) This exercise is to give the student practice at computing sums and products, as well as converting between mathematical notation of sums and products, and *Maple* code. Parts (e.) and (f.) are familiar taylor series that the student should recognise from first year mathematics, and which *Maple* should immediately recognise and simplify. Parts (g.) through (h.) require the student to interpret the ellipsis notation.

Typo: in parts (e.) and (f.) the dummy variable printed on the big sigma should be n, but is printed as k.

(11) This exercise is part practice with function creation, and part binomial experimentation. Note that the student will need to use *Maple*'s help files to find the function that computes the binomial coefficients.

Note that the answer to part (d.) will differ based on whether the students use the inbuilt function, or write their own. In the former case the result will be $(x/y + 1)^N y^N$, and in the latter it will merely be $(x + y)^N$. It is for this reason that the student is instructed to perform the tasks twice. Some thought as to why these are equivalent might be needed by the student.

Typo: Part the first sentence of part (d.) should read *Ask Maple to evaluate* $g(N)$. (The function should he g, instead of f).

(12) This exercise is intended to give the student a mathematical definition (or concept) they have likely not seen before, and leads them through an exploration of the concept. Note that the student is expected to extrapolate the meaning of hexagonal numbers from the explanation of triangular, square, and pentagonal numbers given.

(13) This exercise is to show the student a stumbling block with dummy variables in the sum function. Note that the dummy variables are in the **sum** and **Sum** functions appear to be in the global scope. That is, they do not appear isolated within the function itself. As such, the commands become confused if their dummy variable is already assigned somehow. Note that this assignment might be via another dummy variable, as with the **seq** function being combined with the **sum** function in part (c.).

(14) This exercise is an introduction to the **map** function, which applies a function to each element of an expression. For the purposes of this exercise we have used only lists, but the same concepts we see in (Ex 8.) regarding **op**, **nops**, and **numboccur** apply here.

Note that parts (a.) through (f.) are meant to be demonstrative, and should give the student clues about how to solve parts (g.) through (i.). Students should also look **map** up in the help files.

(15) This exercise demonstrates some more advanced usage of **for** loops. They are shown using incrementation in steps other than 1, including variable assignment from arbitrary lists. Also, the **while** keyword is introduced.

Note that parts (a.) through (f.) are meant to be demonstrative, and should give the student clues about how to solve parts (g.) through (i.).

(16) This exercise is intended to clarify the difference between a **local** and a **global** variable. The student is expected to recognise that the function modifies all of the variables a,b,G, and H. The student should recognise that the modifications to variables a and b do not affect the variables by the same name in the main worksheet, whereas the changes the procedure makes to variables G and H do affect the variables with the same name in the main worksheet. Students who have seen scoping in other languages before should recognise this concept. Other students may well need alternative explanations.

(17) This exercise is to give the student practice in writing procedures. Part (a.) requires the student to realise that there is a possibility of double-counting multiples of 35. Some students will not realise this, and will unknowingly produce an incorrect solution, even if they do test their procedure against some hand calculations as the exercise suggests.

Part (b.) requires the student to create a function of multiple variables. The procedure the student creates will need to accept initial conditions as procedure arguments. The student will also need to find some way to accept an argument telling the procedure how many terms of the sequence to print. This will most likely be a third argument to the procedure, however an enterprising student could very well have their procedure return a new procedure which prints terms of the sequence based on an argument.

(18) This exercise is to get the student thinking about nesting. The student will need to recognise the two recursive patterns in the triangle (going down the rows, and going along the rows). Part (b.) is to reinforce the idea of writing functions or procedures to allow easy repetition of similar computations.

(19) This exercise illustrates the recursive call limit, and how to bypass it with the *remember* **option**. It also explains the comment at the bottom of p39 about the difficulty of finding an argument to the Fibonacci number calculator that will cause the function to take approximately a second to complete its computation.

In order to avoid unterminated loops appearing as part of an infinite backwards recursion, *Maple* limits the number of recursive calls which can happen before it cancels the computation, and issues an error. Some computations simply use too many recursions. Using the *remember* **option** allows us to perform these computations in stages. The student should notice that none of *fib*(6000), *fib*(4000), or *fib*(3000) will compute immediately, but that *fib*(4000) will successfully compute after the computation of *fib*(2000). Similarly, *fib*(6000) will successfully compute after *fib*(4000), but not before.

Note that the execution times are quite quick; all should be well less than a second. This, combined with the observation from the text regarding the measured execution times of this recursive procedure, should explain why we could not find a single computation which took approximately a second.

Warning: in order for subparts (i.) through (iii.) of part (a.) to properly fail, they must be executed when the function has only the initial values in its remember table. This will usually be immediately after defining the function, or after the use of the **forget** function. If the student does some experimentation or testing of the *fib* function before attempting the question, they may not see the expected result.

***Maple* Version Discrepancies:** note that different versions of *Maple* are capable of different amounts of recursive calls before giving an error. Discrepancies can even exist between the GUI interface and the console interface. Some trial and error might be needed to find arguments for the *fib* function which exhibit the behaviour described herein.

(20) This exercise is simply to illustrate that the **%** operator does not work as one might expect with arrow notation functions, and that the **unapply** function should be used instead.

(21) This exercise is to give the student practice at induction. It revisits arithmetic progressions, and has the student perform induction on entire families of arithmetic progressions at once.

(22) This exercise demonstrates the difference between decimal approximations and rational numbers, even if the two ought to represent the same number on paper. The student should find that the loop will not terminate, and that x_i always has 10 digits of precision. This is despite the fact that the loop terminates if we instead begin with the rational number 123456789/10000000000. *Maple* does not consider 0.123456789 to be the exact rational number 123456789/10000000000.

(23) This exercise shows an interesting property of some finite continued fractions related to the continued fraction of the golden ration. The student should notice that each of the continued fraction gives a ratio of successive Fibonacci numbers. This is related to the result $\lim_{n->\infty} f_{n+1}/f_n = \phi$ where f_i is the ith Fibonacci number, and ϕ is the golden ratio.

(24) This exercise is mostly practice with the concept of continued fractions, with a small amount of exploration. None of the numbers appear to have any discernible pattern. The student may think they see a pattern if they compute only few terms of the continued fraction, and so should be encouraged to test their ideas by computing more terms to see if the pattern persists.

(25) This exercise is challenging. It provides practice in using the **rsolve** command, and also reinforces the idea of testing results given by *Maple*. Note that $f(n)$ is an arbitrary function.

The student will likely be unable to solve such a recurrence by hand, but is expected to use basic principles of recurrence relations to test the result given by the **rsolve**. The student likely will not be able to verify the answer in full generality, even with the aid of *Maple*. Instead, the student should choose some pairs of f and p. In some simple cases the result can be easily and directly verified. For others, it is suggested that the student compute the first 20 or so terms of the recurrence using both the definition of the recurrence relation and the expression given by **rsolve**. These values may be compared.

(26) This exercise is simply practice in using **rsolve**. The student should be able to verify all solutions by substitution into the recurrence relation definition.

(27) This question expects the student to find a way to describe the problem as a recurrence relation. Once this is done, then **rsolve** is all that is needed to solve the problem. The solution is a Fibonacci-like relation. If the student finds an alternate (but still correct) solution that does not involve recurrence relations, then more power to them. The author is not aware of such a solution.

(28) This exercise has the student measure execution times of the Sieve of Eratosthenes, and use this data to predict future execution times. Note that the sieve, as implemented, does not have a linear execution complexity, although it is still very efficient. Of particular note is the slowness of the part of the implementation that collates the primes into a sequence. This technique of "growing" a list one element of a time produces a new list at each step, with the current elements being copied to the new list.

There is much experimentation that can be done in this exercise. Many different implementations of functions which compute primes less than or equal to a given bound can be tried, measured and compared. This is a good time for the student to try plotting sequences, if they are familiar with the plotting commands.

Calculus. For most of the exercises in this section, judicious use of the **plot** command can help the student visualise parts of the problem. This is encouraged for all exercises. This should be considered a standing recommendation and we will not explicitly mention this for every exercise.

(1) This exercise is simply to introduce the student to the various student "tutors" and packages that come bundled with *Maple*. These tutors are interactive, and are designed to help students with typical mathematics found in high-school, and university.

Note that, in practice, the author has found that students sometimes think that they are expected to use these tutors for subsequent exercises. This is not the intention, however if the student is not discouraged from doing so if they can make good use of the tools. Such students might need to be encouraged to try and use other techniques when the tutors prove insufficient to the task of solving later exercises.

(2) This exercise is intended to give the student practice in using the **plot** command. For most of these plots, the student is expected to spend some time thinking about sensible plot ranges, and to make sure they show as much information as possible with their plots. Students are discouraged from simply typing the appropriate plot command, and moving on.

    Note that parts (e.) and (f.) produce misleading or unexpected plots in *Maple* version 12. The plot for part (e.) puts the horizontal axis on the line $y = 1$ instead of the more usual $y = 0$. The plot for part (f.) produces a "zig-zaggy" plot showing the decimal fluctuation in the low-order digits of the decimal approximations of the sample points. Close attention to the scale of the vertical axis should illuminate these errors should they show up. The intent of these parts was to reinforce careful attention on the part of the student. Some later versions of *Maple* no longer make these errors.

(3) This exercise is practice in computing limits. The student will need to consider both left and right limits of undefined points. Plots can help, but basic calculus skills should not be neglected.

(4) This exercise is demonstrative of a common mistake in first year. The student should find a value of x for which the second derivative of the function is 0, however this point is not an inflection point. A plot of the function should demonstrate this fact nicely.

(5) This question is a minor challenge, and also practice in using the integration tools package. The student will need to recognise the original integral during the course of the computation, however the integral will likely be using a different dummy variable in this case.

(6) This exercise is practice in performing optimisation computations. The student will need to remember the that the length of an arc is proportional to its angle in radians. Once this is realised, the radius of the circle at the top of the cone, and the height of the cone can be computed, and hence the volume of the cone. Judicious use of functions, as demonstrated in §2.2.1 should help the student keep everything reasonably neat and easier to follow. The student will need to be discriminating when evaluating the points at which the first derivative vanishes. Checking these points by substituting them into the first derivative likely require giving *Maple* assumptions about some variables so that it may correctly simplify the resulting expressions.

    Note that the radius R is entirely arbitrary, and does not affect the final result. This is interesting, but may be missed by the student.

(7) This exercise is practice in numeric evaluation of integrals, and also in use of inverse symbolic computation. Some of these integrals will be able to be performed automatically by *Maple*, and some will need to be numerically evaluated and identified via inverse symbolic computation

Note that messy polylogarithms can be banished with the **simplify** command. If the student tries to numerically evaluate these polylogarithms (without simplifying them), they will get a complex answer (although the imaginary part of this number will be negligible). This may alternatively be avoided by forcing *Maple* to perform numeric integration using the *numeric* argument to the **int** function.

At least one of the integrals will produce a decimal approximation which *Maple*'s **identify** function will not automatically identify. The student is expected to spend some time looking through the help files for the **identify** function. These files are advanced and confusing, but some persistence should eventually yield a list of constants that several steps of the function use. Some of the constants explicitly listed by the exercise are not in this list, and the student must find the option that allows them to add such constants. Once this is done, the results follow. The student should test these results to extra precision levels to boost their confidence in the veracity of the identification.

Note that (e.) and (f.) are highly unlikely to be able to be correctly identified by the student. Shortly after the publication of the text, a closed form was found for the integral in part (e.). These two were included solely for interest, and to remind the student that unanswered questions exist.

(8) This exercise provides practice in using the **dsolve** function, and in verifying results. Verification is as simple as substituting the solution back into the differential equation. Note that the constant that is referred to is the constant of integration which will appear in the solution of the differential equation. These equations should be able to be solved by hand by any student taking this course.

Plotting the solution curves is a nice visualising touch. Students may simply plot the curves using the **plot** command manually. However, the enterprising student is encouraged to attempt to work out the **DePlot** function, as suggested by the exercise itself. Doing so should produce both a vector field, as well as the requested solution curves.

(9) This exercise introduces the Bessel functions to the student. The student is likely incapable of solving these differential equations by hand, but they may verify the solution given to them by the **dsolve** function.

(10) This exercise consists of two challenges in plotting with polar coordinates.

Part (a.) is a cute exercise in drawing Pac-Man. The student is required to use the plotting techniques demonstrated in §2.2.5 and some lateral thinking to produce a plot of pPa. Only the basic outline is needed; the student need not fill the interior, or add the eye (however, if the student succeeds in this, then all the better).

Part (b.) requires the student to find the polar equation of a particular circle. Note that this question is not asking for the polar equation of a general circle with a general centre. The student will likely need to use the standard identities for converting between cartesian and polar coordinates. The resulting equation is messy, but with *Maple* to handle the algebra, is reasonably straightforward.

(11) This exercise is practice in using the techniques of computing volumes of rotation as seen in §2.3.2. The interesting result hinted at in part (a.) is that the result should be exactly the same as the volume of rotation obtained by rotating area underneath the same curve (i.e., between the curve and the x-axis) around the z-axis. The paraboloid effectively cuts the volume of a cylinder in half.

(12) This exercise is practice in finding and classifying critical points in functions of two variables. Note that in part (b.) that the solutions produced by **solve** for when the partial derivatives vanish that involve the **RootOf** function. In particular, we see something along the lines of $\mathit{RootOf}(-1 + 2z^2)$ which is *Maple*'s way of representing the solution set of $-1 + 2z^2 = 0$. The student will need to look in the help files to find this information. A plot of the function should suggest the existence of 4 or 5 critical points, instead of the 2 which are suggested by the result of the **solve** function.

(13) This exercise provides more practice in verifying differential equations; partial differential equations this case. The solutions are given and the student is required only to verify them by substitution into the partial differential equation. The enterprising student might find and use the **pdsolve** and find that a solution, or the student might simply notice that all the solutions presented are a function of $x - y$. In any event, the student should verify their general solution by substitution.

(14) This exercise is a demonstration of a case in which Clairaut's theorem fails. Students will likely find this challenging. In order to see the difference in the second derivatives, the limit definition of definitions must be used. Fortunately, this definition need only be used for the case of the origin.

Note that the use of **piecewise** to define the function f will not be illuminating, as *Maple* will differentiate it part by part, with no regard to the limits. That is, *Maple* will give an incorrect answer for the partial derivatives. The author recommends defining f to be the function $(x, y) \mapsto (xy(x^2 - y^2)) \,/\, (x^2 + y^2)$ and to explicitly define $f(0, 0)$ to be equal to 0 (in much the same way initial values are assigned for recursive functions in Chapter 1).

(15) This exercise provides practice in interpreting areas as iterated integrals, and in computing iterated integrals. Note that parts (c.) and (d.) require expressions in the variable y to be part of the range argument to the **int** command. The student will not have seen this before, but if they simply attempt to write the expressions into the range argument, they should find that it works just fine.

Linear Algebra. Note that, with the exception of (Ex 1.), the student may use whichever method they prefer when entering matrices or vectors.

(1) This question is simply practice in entering matrices and vectors into *Maple*. The questions are prescriptive as to method used so as to make sure the student practices each method. This is the only question which prescribes methods of entering matrices and vectors.

    Note that for part (c.) the student is required to recognise a family of matrices by their pattern, and to write a function whose argument (or arguments) are the parameters for that family. The student should be able to compute specific members of the relevant family using the function they create. For example, the case where $n = 7$ of part (i.), or the case where $n = 6, m = 8$ for parts (ii.) or (iii.).

(2) This exercise provides practice with matrix and vector arithmetic in *Maple*. Parts (b.) and (c.) have the student recall results from first-year linear algebra, so that the examples are not quite so contrived, and require a little problem solving.

(3) This exercise is practice with linear systems. Plotting the systems is a handy way to visualise the system, whether it has a solution or not. In some cases, the visualisation of how a system fails to have a solution can be illuminating for the student who is weaker in first-year linear algebra.

    To express the solutions of the systems in part (b.), the reader is required to understand that solving a system of simultaneous linear equations is equivalent to solving the matrix equation $Ax = b$. If the student understands that this, in turn, is equivalent to solving questions of linear combinations of the column vectors of A, then all the better.

(4) This exercise requires the student to see a pattern in a family of matrices, and reinforces the techniques from (Ex 1.) part (c.). In this case there are initially apparently two related patterns, depending on whether n is odd or even, however closer inspection should yield a single pattern which works in both cases.

(5) This exercise is simply practice in finding inverses through row reduction, and manipulation of matrices in *Maple*. Note that this is not a sensible way of finding matrix inverses in *Maple*. The student is required to understand the proof from §3.1.3 sufficiently to realise that if the reduced row echelon form of a matrix is no the identity matrix, then it is not an invertible matrix.

    The student need not use elementary matrices to solve this question (and, in fact, is encouraged not to). Simply constructing the augmented matrix using angle bracket notation and the | operator, and using the **ReducedRowEchelonForm** function is sufficient.

(6) This exercise tests whether the student understands the theorem from §3.1.3 regarding equivalent properties of an invertible matrix. The student who does not understand this theorem should still be capable of answering the question, but will take far longer do to so; checking each property separately.

(7) This exercise is intended to reinforce the concepts introduced in §3.1.3. Note that this is a horrible way to find inverses from a practical point of view, however it should reinforce the ideas involved in invertible matrices being built out of elementary matrices. The student need not actually use elementary matrices; the **RowOperation** function will suffice.

Note that the student is encouraged reverse the order of the operations, and apply them to the identity matrix to make sure they get their original matrix back.

(8) This exercise reinforces the idea finding linear combinations by solving linear systems. For part (b.) the student will need to be able to convert between the polynomial spaces, and $\mathbb{R}^n$.

(9) This exercise reinforces the idea that questions of linear dependence, and questions of linear combinations are one in the same. The student is expected to use information gleaned from their solution to (Ex 8.) to reduce the work required for this exercise.

The student who recalls that a set of vectors are linearly dependent if one can be written as a combination of the others may use this to rule out some sets of vectors or polynomials. However such a student should realise that any inconsistent systems from (Ex 8.) will require further checking. The weaker student might simply check each of the sets of vectors from (Ex 8.).

(10) This exercise is a challenge to the student to extend the techniques used in the text to form an isomorphism between $P_n(\mathbb{F})$ and $\mathbb{F}^n$. In this case the student is expected to realise that the matrix space $M_n(\mathbb{R})$ is n^2-dimensional and to produce suitable functions for conversion between elements of $M_n(\mathbb{R})$ and $\mathbb{R}^{n^2}$.

(11) This exercise is a challenge. Part (a.) is largely conceptual, but requires some thought. Note that a formal proof is not necessary, a good "hand-waving" argument should suffice.

Part (b.) requires the student to extrapolate the techniques from §3.3.2 for finding a matrix representation of a linear transformation, however in this case they will not have a function implemented in *Maple* with which to rely upon. Thought on the part of the student should yield that rotation in, say, the xz plane will not affect the y component of a vector (and similarly for the other planes stipulated).

The remainder of the question is a challenge.

(12) This exercise provides practice in computing eigenvectors and eigenvectors in *Maple*.

(13) This exercise requires the student to do some problem solving. Part (a.) requires the student to realise that powers of a diagonal matrix remain diagonal, and to use that realisation to solve the problem. Part (b) requires the student to recall from §3.3.4 that powers of diagonalisable matrices rely only on computing powers of the diagonal "part" of that matrix (i.e., the D where $M = P^{-1}DP$).

(14) This exercise provides practice in diagonalising matrices.

An Introduction to Modern Mathematical Computing with *Mathematica*

This document is intended to act as a guide to the exercises in the book “An Introduction to Modern Mathematical Computing with *Mathematica*”. It is primarily intended for instructors, lecturers, tutors or the like, and outlines the intent of the exercises, as well as any technical points, stumbling blocks, or other significant point the student or reader might miss. Students should feel free to read the contents of this document, but should be warned that the material is aimed at a higher level than the books are aimed at.

For the duration of this document we will refer to the person attempting the exercises as *the student*, although we acknowledge that not everybody attempting the exercises from the text will necessarily be a student. The book itself, and the exercises therein, was primarily written as a text to accompany a second year university course, and as such the exercises were written with teaching students in mind.

Number Theory.

(1) The goal of this exercise is to give students practice in entering basic expressions into *Mathematica*, as well as in using the `N` function. Note that part (h.) may potentially lead to a discussion as to the precedence of taking powers. That is

$$2^{2^{2^{2^{2}}}} = 2^{\left(2^{\left(2^{(2^2)}\right)}\right)} \neq \left(\left(\left(2^2\right)^2\right)^2\right)^2 = 2^{2\cdot 2\cdot 2\cdot 2}$$

Furthermore, *Mathematica* gives an exact integer answer consisting of the entire 19,729 digits, but rounds rounds the floating point approximation to 16 significant figures (or however many significant figures the student specifies).

(2) This exercise is to show the student how *Mathematica* (or, specifically, the `N` function) handles digit precision of decimal approximations. Note that parts (a.) through (d.) are meant to be demonstrative, and should give the user clues about how to solve parts (f.) through (h.).

Also, note that *Mathematica* does not have a way of setting precision for an entire notebook (or, at least, the author has thus far failed to find one). Precision is handled on a per-number basis, with the most appropriate precision being selected automatically for most numeric computations.

(3) This exercise demonstrates two possible stumbling blocks. Firstly (part (a.)) that some variable names are protected in *Mathematica*. Common names the students will likely end up wanting to use themselves will be `I` (the imaginary unit) and `D` (the differential operator). This exercise should introduce the idea that some variable names are already taken, and can’t be used.

Note that *Mathematica* provides the capability to remove or grant protection via the **`Unprotect`** and **`Protect`** functions respectively. Doing so should not be necessary in the context of the book, or its exercises, and so these functions are not mentioned at all there.

Secondly, (part (b.)), demonstrates a common misunderstanding with the `%` operator. Specifically, the `%` operator represents the result of the *most recently* performed computation. This exercise demonstrates that the most recently performed computation is not necessarily the computation on the previous line of the workbook.

(4) This exercise has the student learn a little about the `Array` function, and also introduces pure functions. The student is expected to consult the Documentation Center regarding both topics. Note that the `Array` function is used in §1.2.6, and a brief explanation of it can be found there.

`Array` is similar to `Table`, but differs in that it must be initialised by a function (its first argument) and that it does not use the iterator notation. On the whole, the author finds `Array` to be less flexible than `Table`, but useful in some specific circumstances.

Pure functions allow the expression of functions without having to assign a name to the function, and without using *Mathematica*'s pattern matching functionality. Note that pure functions may be assigned using the `&` operator, as shown in this exercise, or equivalently using the `Function` function.

(5) This exercise is to show the student some of the more technical capabilities regarding iterator notation. Note that parts (a.) through (d.) are meant to be demonstrative, and should give the student clues about how to solve parts (e.) and (f.).

Note that the intent for part (f.) was to have the student to solve the exercise using a list inside iterator (i.e., with `Table[1/k, {k,{1,2,4,7,11,16}}]` or something similar). Some students notice that the denominators exhibit a pattern where first 1 is added, then 2, then 3, and so on. Some careful mathematics should show that the kth term of this sequence is $1 + \frac{n\,(n-1)}{2}$ and so the problem may be solved by `Table[2/(k^2-k+2), {k,1,6}]`.

(6) This exercise is to show the student how a list may be indexed in reverse. Note that the failures are produced because the ranges are backwards. For example, part (g.) is equivalent to asking for `L[[10;;8]]`, which is not a valid range because the left hand side is larger than the right hand side.

(7) This exercise is to demonstrate the `Length` and `Count` functions. The student is expected to look at the Documentation Center to work out how to solve the challenge at the end of the exercise. Note that the `Count` function can make use of *Mathematica*'s pattern matching for its second argument if needed, although this exercise does not demonstrate this.

Also note that the `Count` function is capable of counting patterns that appear as sub-expressions of any *Mathematica* expression, although we do not demonstrate that functionality in this exercise. The **Length** function is also capable of operating on arbitrary expressions, although its behaviour in this case is highly technical and beyond the scope of the book and these exercises.

(8) This exercise is related to the *further exploration* of normal numbers (§1.5, p65). If the student looks in the Documentation Center they should find a function which will produce the desired result more or less directly. If this function is not found, the student may write their own such function. A loop consisting of truncation, multiplication by 10 and subtraction should suffice. The student in this situation ought to be careful to make sure they begin the procedure with enough precision, or rounding errors might creep in.

(9) This exercise is to give the student practice at computing sums and products, as well as converting between mathematical notation of sums and products, and *Mathematica* code. Parts (e.) and (f.) are familiar taylor series that the student should recognise from first year mathematics, and which *Mathematica* should immediately recognise and simplify. Parts (g.) through (h.) require the student to interpret the ellipsis notation.

Typo: in parts (e.) and (f.) the dummy variable printed on the big sigma should be n, but is printed as k.

(10) This exercise is part practice with function creation, and part binomial experimentation. Note that the student will need to use *Mathematica*'s Documentation Center to find the function that computes the binomial coefficients.

Typo: Part the first sentence of part (d.) should read *Ask Mathematica to evaluate* $g(N)$. (The function should he g, instead of f).

(11) This exercise explores a relationship between sums and products. Note that *Mathematica* is unable to compute these sums symbolically, and consequently the student will need to use numeric approximations. The student should find that the value they obtain for part (c.), when raised as the power of e, should be the same as the value they obtain for part (a.).

In order to justify the relationship in general, limits of partial sums will need to be used. This justification may be proved on paper or in *Mathematica* as the student prefers.

Typo: part (d.) should read "*Verify that the relationship described at the beginning of this exercise holds for the sum and product from parts (a.) and (c.) respectively.*"

(12) This exercise is intended to give the student a mathematical definition (or concept) they have likely not seen before, and leads them through an exploration of the concept. Note that the student is expected to extrapolate the meaning of hexagonal numbers from the explanation of triangular, square, and pentagonal numbers given.

(13) This exercise is an introduction to the `Map` function, and the operator, and a revisiting of pure functions. The `Map` function will apply a function to each element of an expression. For the purposes of this exercise we have used only lists, but much like the `Count` function introduced in (Ex 7.), the `Map` function will work on any *Mathematica* expression.

Note that parts (a.) through (f.) are meant to be demonstrative, and should give the student clues about how to solve parts (g.) through (i.). Students are also encouraged look up the `Map` function in the Documentation Center.

(14) This exercise demonstrates some advanced loops. The advanced iterator notations from (Ex 5) are used with `Do` loops, and `While` loops are introduced. Pure functions are also touched on once more.

Note that parts (a.) through (f.) are meant to be demonstrative, and should give the student clues about how to solve parts (g.) through (i.).

(15) This exercise is to give the student practice in writing procedures. Part (a.) requires the student to realise that there is a possibility of double-counting multiples of 35. Some students will not realise this, and will unknowingly produce an incorrect solution, even if they do test their procedure against some hand calculations as the exercise suggests.

Part (b.) requires the student to create a function of multiple variables. The procedure the student creates will need to accept initial conditions as procedure arguments. The student will also need to find some way to accept an argument telling the procedure how many terms of the sequence to print. This will most likely be a third argument to the procedure, however an enterprising student could very well have their procedure return a new procedure which prints terms of the sequence based on an argument.

(16) This exercise is to get the student thinking about nesting. The student will need to recognise the two recursive patterns in the triangle (going down the rows, and going along the rows). Part (b.) is to reinforce the idea of writing functions or procedures to allow easy repetition of similar computations.

(17) This exercise introduces the `NSum` function, which behaves much like the `Sum` function, except that it performs purely numeric computations.. The student is required to use the Documentation Center in order to solve the exercise. Timing methods are expected to be employed to see whether the techniques from §1.2.2 (involving the `Sum` function as an argument to the `N` function) are slower than using `Nsum` directly.

(18) This question challenges the student to find an iterative way to compute Fibonacci numbers. An iterative method should avoid the problem described in §1.2.6 p48 regarding the computation of Fibonacci numbers past the 7,000th. Note that the student should beware of the problems inherent in timing recursive functions which remember previous computed values, as discussed in §1.2.6.

(19) This exercise is simply to illustrate that the `%` operator does not work as one might expect when defining functions, and that the `Evaluate` function should be used instead.

(20) This exercise is to give the student practice at induction. It revisits arithmetic progressions, and has the student perform induction on entire families of arithmetic progressions at once.

(21) This exercise demonstrates the difference between decimal approximations and rational numbers, even if the two ought to represent the same number on paper. The student should find that the loop will not terminate, and that `x[i]` always have approximate the same amount of precision. This is despite the fact that the loop terminates if we instead begin with the rational number 123456789/10000000000. *Mathematica* does not consider 0.123456789 to be the exact rational number 123456789/10000000000.

(22) This exercise shows an interesting property of some finite continued fractions related to the continued fraction of the golden ration. The student should notice that each of the continued fraction gives a ratio of successive Fibonacci numbers. This is related to the result $\lim_{n->\infty} f_{n+1}/f_n = \phi$ where f_i is the ith Fibonacci number, and ϕ is the golden ratio.

(23) This exercise is mostly practice with the concept of continued fractions, with a small amount of exploration. None of the numbers appear to have any discernible pattern. The student may think they see a pattern if they compute only few terms of the continued fraction, and so should be encouraged to test their ideas by computing more terms to see if the pattern persists.

(24) This exercise is challenging. It provides practice in using the **RSolve** function, and also reinforces the idea of testing results given by *Mathematica*. Note that $f(n)$ is an arbitrary function.

The student will likely be unable to solve such a recurrence by hand, but is expected to use basic principles of recurrence relations to test the result given by the **RSolve**. The student likely will not be able to verify the answer in full generality, even with the aid of *Mathematica*. Instead, the student should choose some pairs of f and p. In some simple cases the result can be easily and directly verified. For others, it is suggested that the student compute the first 20 or so terms of the recurrence using both the definition of the recurrence relation and the expression given by **RSolve**. These values may be compared.

(25) This exercise is simply practice in using **RSolve**. The student should be able to verify all solutions by substitution into the recurrence relation definition.

(26) This question expects the student to find a way to describe the problem as a recurrence relation. Once this is done, then **RSolve** is all that is needed to solve the problem. The solution is a Fibonacci-like relation. If the student finds an alternate (but still correct) solution that does not involve recurrence relations, then more power to them. The author is not aware of such a solution.

(27) This exercise has the student measure execution times of the Sieve of Eratosthenes, and use this data to predict future execution times. Note that the sieve, as implemented, does not have a linear execution complexity, although it is still very efficient.

There is much experimentation that can be done in this exercise. Many different implementations of functions which compute primes less than or equal to a given bound can be tried, measured and compared. This is a good time for the student to try plotting sequences, if they are familiar with the plotting commands (or if they are adventurous).

Typo: Part (b.) should be ignored. There is no part to the implementation of the sieve from §1.3.4 that collates the primes into a list, and so part (b.) cannot be done.

Calculus. For most of the exercises in this section, judicious use of the **plot** command can help the student visualise parts of the problem. This is encouraged for all exercises. This should be considered a standing recommendation and we will not explicitly mention this for every exercise.

(1) This exercise is intended to give the student practice in using the `Plot` command. For most of these plots, the student is expected to spend some time thinking about sensible plot ranges, and to make sure they show as much information as possible with their plots. Students are discouraged from simply typing the appropriate plot command, and moving on.

(2)

(3) This exercise is practice in computing limits. The student will need to consider both left and right limits of undefined points. Plots can help, but basic calculus skills should not be neglected.

(4) This exercise is demonstrative of a common mistake in first year. The student should find a value of x for which the second derivative of the function is 0, however this point is not an inflection point. A plot of the function should demonstrate this fact nicely.

(5) This exercise is practice in performing optimisation computations. The student will need to remember the that the length of an arc is proportional to its angle in radians. Once this is realised, the radius of the circle at the top of the cone, and the height of the cone can be computed, and hence the volume of the cone. Judicious use of functions, as demonstrated in §2.2.1 should help the student keep everything reasonably neat and easier to follow. The student will need to be discriminating when evaluating the points at which the first derivative vanishes.

Note that the radius R is entirely arbitrary, and does not affect the final result. This is interesting, but may be missed by the student.

(6) This exercise is practice in numeric evaluation of integrals, and also in use of inverse symbolic computation. Some of these integrals will be able to be performed automatically by *Mathematica*, and some will need to be numerically evaluated and identified via inverse symbolic computation. Numeric integration can be performed by feeding the output of the `Int` function into the `N` function, or by directly using the `NIntegrate` function.

Note that (e.) and (f.) are highly unlikely to be able to be correctly identified by the student. Shortly before the publication of the text, a closed form was found for the integral in part (e.). These two were included solely for interest, and to remind the student that unanswered questions exist.

(7) This exercise provides practice in using the **`DSolve`** function, and in verifying results. Verification is as simple as substituting the solution back into the differential equation. Note that the constant that is referred to is the constant of integration which will appear in the solution of the differential equation. These equations should be able to be solved by hand by any student taking this course.

(8) This exercise introduces the Bessel functions to the student. The student is likely incapable of solving these differential equations by hand, but they may verify the solution given to them by the **`DSolve`** function.

(9) This exercise consists of two challenges in plotting with polar coordinates.

Part (a.) is a cute exercise in drawing Pac-Man. The student is required to use the plotting techniques demonstrated in §2.2.5 and some lateral thinking to produce a plot of PPac-Man. Only the basic outline is needed; the student need not fill the interior, or add the eye (however, if the student succeeds in this, then all the better).

Part (b.) requires the student to find the polar equation of a particular circle. Note that this question is not asking for the polar equation of a general circle with a general centre. The student will likely need to use the standard identities for converting between cartesian and polar coordinates. The resulting equation is messy, but with *Mathematica* to handle the algebra, is reasonably straightforward.

(10) This exercise is practice in using the techniques of computing volumes of rotation as seen in §2.3.2. The interesting result hinted at in part (a.) is that the result should be exactly the same as the volume of rotation obtained by rotating area underneath the same curve (i.e., between the curve and the x-axis) around the z-axis. The paraboloid effectively cuts the volume of a cylinder in half.

(11) This exercise is practice in finding and classifying critical points in functions of two variables. Note that in part (b.) that the **`Solve`** function might output a warning when computing where the partial derivatives vanish. Nonetheless, a list consisting of five solutions should be returned. A plot of the function should suggest the existence of 4 or 5 critical points in keeping with the solutions provided by **`Solve`**.

(12) This exercise provides more practice in verifying differential equations; partial differential equations this case. The solutions are given and the student is required only to verify them by substitution into the partial differential equation. The enterprising student might find a function in the Documentation Center capable of solving partial differential equations, or the student might simply notice that all the solutions presented are a function of $x - y$. In any event, the student should verify their general solution by substitution.

(13) This exercise is a demonstration of a case in which Clairaut's theorem fails. Students will likely find this challenging. In order to see the difference in the second derivatives, the limit definition of definitions must be used. Fortunately, this definition need only be used for the case of the origin. Note that the use of

`Piecewise` to define the function f will not be illuminating, as *Mathematica* will differentiate it part by part, with no regard to the limits. That is, *Mathematica* will give an incorrect answer for the partial derivatives. The author recommends defining f to be the function $(x, y) \mapsto (xy(x^2 - y^2)) \,/\, (x^2 + y^2)$ and to explicitly define $f(0, 0)$ to be equal to 0 (in much the same way initial values are assigned for recursive functions in Chapter 1).

(14) This exercise provides practice in interpreting areas as iterated integrals, and in computing iterated integrals. Note that parts (c.) and (d.) require expressions in the variable y to be part of the range argument to the `Integrate` command. The student will not have seen this before, but if they simply attempt to write the expressions into the range argument, they should find that it works just fine. Care should be taken to make sure the multiple iterators are presented in the correct order.

Linear Algebra. Note that, with the exception of (Ex 1.), the student may use whichever method they prefer when entering matrices or vectors.

(1) This question is simply practice in entering matrices and vectors into *Mathematica*. The questions are prescriptive as to method used so as to make sure the student practices each method. This is the only question which prescribes methods of entering matrices and vectors.

Note that for part (c.) the student is required to recognise a family of matrices by their pattern, and to write a function whose argument (or arguments) are the parameters for that family. The student should be able to compute specific members of the relevant family using the function they create. For example, the case where $n = 7$ of part (i.), or the case where $n = 6, m = 8$ for parts (ii.) or (iii.).

Typo: Part (a.) should read "*Create the following vectors and matrices using brace notation {}.*"

(2) This exercise provides practice with matrix and vector arithmetic in *Mathematica*. Parts (b.) and (c.) have the student recall results from first-year linear algebra, so that the examples are not quite so contrived, and require a little problem solving.

(3) This exercise is practice with linear systems. Plotting the systems is a handy way to visualise the system, whether it has a solution or not. In some cases, the visualisation of how a system fails to have a solution can be illuminating for the student who is weaker in first-year linear algebra.

To express the solutions of the systems in part (b.), the reader is required to understand that solving a system of simultaneous linear equations is equivalent to solving the matrix equation $Ax = b$. If the student understands that this, in turn, is equivalent to solving questions of linear combinations of the column vectors of A, then all the better.

(4) This exercise requires the student to see a pattern in a family of matrices, and reinforces the techniques from (Ex 1.) part (c.). In this case there are initially apparently two related patterns, depending on whether n is odd or even, however closer inspection should yield a single pattern which works in both cases.

(5) This exercise is simply practice in finding inverses through row reduction, and manipulation of matrices in *Mathematica*. Note that this is not a sensible way of finding matrix inverses in *Mathematica*. The student is required to understand the proof from §3.1.3 sufficiently to realise that if the reduced row echelon form of a matrix is no the identity matrix, then it is not an invertible matrix.

The student need not use elementary matrices to solve this question (and, in fact, is encouraged not to). Simply constructing the augmented matrix and using the `RowReduce` function is sufficient.

(6) This exercise tests whether the student understands the theorem from §3.1.3 regarding equivalent properties of an invertible matrix. The student who does not understand this theorem should still be capable of answering the question, but will take far longer do to so; checking each property separately.

(7) This exercise is intended to reinforce the concepts introduced in §3.1.3. Note that this is a horrible way to find inverses from a practical point of view, however it should reinforce the ideas involved in invertible matrices being built out of elementary matrices. The student need not actually use elementary matrices; the row operation functions defined in §3.1.3 will suffice.

Note that the student is encouraged reverse the order of the operations, and apply them to the identity matrix to make sure they get their original matrix back.

(8) This exercise reinforces the idea finding linear combinations by solving linear systems. For part (b.) the student will need to be able to convert between the polynomial spaces, and $\mathbb{R}^n$.

(9) This exercise reinforces the idea that questions of linear dependence, and questions of linear combinations are one in the same. The student is expected to use information gleaned from their solution to (Ex 8.) to reduce the work required for this exercise.

The student who recalls that a set of vectors are linearly dependent if one can be written as a combination of the others may use this to rule out some sets of vectors or polynomials. However such a student should realise that any inconsistent systems from (Ex 8.) will require further checking. The weaker student might simply check each of the sets of vectors from (Ex 8.).

(10) This exercise is a challenge to the student to extend the techniques used in the text to form an isomorphism between $P_n(\mathbb{F})$ and $\mathbb{F}^n$. In this case the student is expected to realise that the matrix space $M_n(\mathbb{R})$ is n^2-dimensional and to produce suitable functions for conversion between elements of $M_n(\mathbb{R})$ and $\mathbb{R}^{n^2}$.

(11) This exercise is a challenge. Part (a.) is largely conceptual, but requires some thought. Note that a formal proof is not necessary, a good “hand-waving” argument should suffice.

Part (b.) requires the student to extrapolate the techniques from §3.3.2 for finding a matrix representation of a linear transformation, however in this case they will not have a function implemented in *Mathematica* with which to rely upon. Thought on the part of the student should yield that rotation in, say, the xz plane will not affect the y component of a vector (and similarly for the other planes stipulated).

The remainder of the question is a challenge.

(12) This exercise provides practice in computing eigenvectors and eigenvectors in *Mathematica*.

(13) This exercise requires the student to do some problem solving. Part (a.) requires the student to realise that powers of a diagonal matrix remain diagonal, and to use that realisation to solve the problem. Part (b) requires the student to recall from §3.3.4 that powers of diagonalisable matrices rely only on computing powers of the diagonal “part” of that matrix (i.e., the D where $M = P^{-1}DP$).

(14) This exercise provides practice in diagonalising matrices.