MATHEMATICAL INSTITUTE

UNIVERSITY OF OXFORD

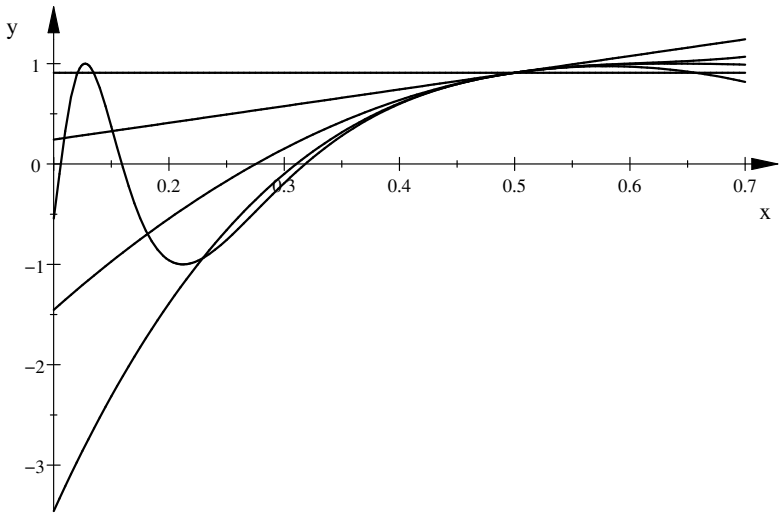
Exploring Mathematics with MuPAD

Students' Guide

Michaelmas Term 2011

by

Colin MacDonald



Version 1.0, October 2010, written by Catherine Wilkins

Acknowledgement to Version 1.0: This course guide has been based on previous guides written for the 'Exploring Mathematics with Maple' course which preceeded this one. I am grateful to all previous authors and to those who assisted them.

Notes for the 2011/2012 Course

The course will be in two parts: Part I in Michaelmas Term Weeks 3-8 and Hilary Term Weeks 1-2, and Part II in Hilary Term Weeks 3-8. For Part I you will be allocated a computer to yourself during scheduled sessions in the Statistics Department, and you may work collaboratively with others in these sessions. None of the work in Part I will be assessed, but instead will act as a foundation enabling you to work individually during Part II. This individual work will be assessed and will count towards your Moderations as described in *Examination Decrees & Regulations*, 2011 and the current *Course Handbook*. (See your college tutor if you have any questions about this aspect of the course.)

The practical sessions will run in fortnightly cycles, each cycle starting on the Monday of an odd week (starting in week 3, Michaelmas Term). You will be allocated a time to attend one practical session during each cycle. The practical sessions are held in the Statistics Department, South Parks Road. During each practical session a demonstrator will be available for help. Problems are provided for each chapter and you should take some time to prepare for each practical session by reading the chapter to be covered in advance.

Week	Chapter Covered
3-4 MT	1
5-6 MT	2
7-8 MT	3
1-2 HT	4

The above table is only a rough guide. You should work through each chapter in order, progressing at a rate of roughly one chapter a fortnight. You need to put in *concentrated* effort to ensure that you make steady progress and that you finish the course by the end of Week 2, Hilary Term. The material covered in Chapter 4 is particularly important for the projects that you will work on in Hilary Term Weeks 3-8. However, the material in this manual is sequential and in order to understand Chapter 4 you will need to have worked methodically through Chapters 1–3 first.

There are two additional chapters that cover MuPAD commands, namely Appendices A and B. The material in Appendix A will be needed for the compulsory first project next term, and may be worked through at any time after you have finished Chapter 1. Appendix B, which contains some useful information on simplifying the output obtained from MuPAD, is appropriate any time after Chapter 3. There are certain rules which apply to the use of computers and these are detailed in Appendix C. A Glossary of commands is provided for you at the end of this manual, in Appendix D.

The course director for this academic year is **Dr Colin MacDonald**. The course demonstrators (who will be in attendance during each practical session) should be able to answer any of your questions; alternatively Dr MacDonald can be contacted by email at **colin.macdonald@maths.ox.ac.uk**.

This manual and any extra course material (such as lecture notes) can be found on the course website
`https://www.maths.ox.ac.uk/courses/course/13447/material`

Contents

Chapter 1

Introduction to the system and to MuPAD

In this chapter you will learn about:

- MuPAD notebooks - opening and saving them;
- using the MuPAD help system;
- using MuPAD as a calculator;
- MuPAD identifiers, constants and functions;
- plotting simple line graphs.

1.1 Introduction to the Statistics Department system

Before you start work on MuPAD, you should ensure that you can log on to the computers in the Statistics Department and open MuPAD successfully. Details of how to do this are provided in a separate handout, and are also found at

`http://www.stats.ox.ac.uk/about_us/it_information/restrictedaccess/undergraduate_matlab_server`

(This address, which can only be accessed from within the University network, should be typed in as one single long line.)

1.1.1 Opening and saving notebooks

This subsection is not important when you are working through this chapter for this first time. It is placed here for completeness and will become useful as you progress through the manual. So either skim-read it now, or leave it out altogether and come back to it later when you need it.

You are advised to save the MuPAD notebooks that you produce as you work through this manual, perhaps saving one for each main section covered (Section 1.3, Section 1.4, etc.). Then you will be able to access the material later on if you need to (this will be very important next term when you work through the projects). To save a notebook, choose the `Save` command on the `File` drop-down menu (or click the 'Save' button on the toolbar). If the notebook has been

saved previously, it will be saved under the existing name. Otherwise, you will be presented with a file browser and asked to choose a file name. Note that MuPAD will append to the filename the ending `.mn` to denote the fact that it is a MuPAD notebook, and not some other kind of file. The first section you are advised to save is Section 1.3 and you will be prompted to do so at the end of that section.

To open an existing notebook, choose the `Open` command on the `File` menu (or click the 'Open' button). You will again be presented with a file browser and asked to choose the file.

1.1.2 Finishing MuPAD

Pull down the File menu, and choose Exit.

1.2 Using MuPAD's built-in help system

Before we introduce you to MuPAD properly, we just want to draw your attention to its extensive and comprehensive built-in help system. One way of accessing this is via the `Help` menu on the toolbar and another is by typing a question mark `?` at the MuPAD prompt, followed by the command you want help with. Say you wanted to search for help on the command '`solve`'. At the prompt type `?solve` to bring up a new window with information on `solve`. Try it now. The scroll bar on the right hand side of the window can be used to move up and down the information. You will probably find the first half of each help page largely irrelevant. The most useful part will be the lower half of the page where examples are given and related commands are shown. Often it is helpful to cut and paste and edit these examples. One of the most important functions of the help pages is to indicate exactly what arguments the command should be supplied with.

Throughout this course you will be encouraged to use MuPAD's help facility to learn more about the commands you are introduced to. You will be asked to access various help pages at several stages in this manual, but you should do so at any stage if the workings of any particular command are unclear to you.

1.3 Introduction to MuPAD

Finally we introduce MuPAD, which is one of a number of algebraic computing systems. It can be used as a basic calculator, for example to evaluate $\ln 3$ or $\sqrt{2}$ to any number of decimal places (within reason), but its great strength is its ability to manipulate *algebraic* expressions. MuPAD can be used to solve algebraic equations, manipulate matrices and vectors, factorise polynomials and simplify rational expressions. It knows about complex numbers and it can also perform many of the standard operations of analysis such as evaluating limits, derivatives and integrals. It has good 2D and 3D graphics facilities so that, for example, solutions of ordinary differential equations can be found and plotted. It is also a programming language and, by writing and executing MuPAD programs, quite complicated operations can be carried out.

This Michaelmas Term course introduces you to several of these features of MuPAD and it sets the scene for many more. First you will learn how to input simple expressions into MuPAD. You will then look at how to plot simple 2D line graphs and how to solve certain equations both symbolically and numerically. You will then be introduced to some of the basic calculus facilities of MuPAD and you will see how to differentiate and integrate expressions. You will also learn how to use MuPAD to solve ordinary differential equations and simple recurrence relations, as well as how to evaluate sums and limits. In the final chapter of this manual you

will be introduced to some of MuPAD's programming facilities.

Although this manual appears to introduce a lot of MuPAD commands many, such as `plot, sum, diff` and `int`, are almost self-explanatory. A list of the most common commands, giving more information, is contained in the Glossary of commands in Appendix D, and you are encouraged also to use the MuPAD 'help' facility as explained in Section 1.2. In order to become technically adept at applying MuPAD, it is important that you attempt *all* the exercises contained in this manual (except those marked as optional); remember, MuPAD is best learned by actually *using* it to *do* mathematics, and this should be practised as often as possible. Try to incorporate MuPAD into your weekly problem sheets, perhaps using it to check some of your hand-written answers.

In this section we will give a brief introduction to MuPAD, outlining some of its basic features and illustrating them with some short examples. It is assumed that you have read Sections 1.1 and 1.2, and have an open Notebook in front of you. You will need to type in all the commands that you see in this, and subsequent, chapters; in this way you will become increasingly familiar with the basic ideas. The output of each command line is not printed in this manual; after executing each line you should check that the output is as you would expect. If it is not, then you should think about why this might be, and if necessary consult your demonstrator.

1.3.1 General ideas

As a start, note that in MuPAD the prompt is the symbol `[`. Commands are entered to the right of the prompt, with each command ending with either a colon or a semi-colon. If a colon is used, the command is executed but the output is not printed, whereas when a semi-colon is used the output is printed. Once you have finished typing a command line then the return/enter key should be pressed. To illustrate this try the following. At the prompt, type `12/15;` followed by the return/enter key. You should see

```
[  12/15;
```

followed by the answer, 4/5. However, typing

```
[  12/15:
```

suppresses the output so that no output is seen (this is useful, for example, if the output is extensive and not specifically required). If you forget the colon/semi-colon MuPAD will output the result as if you had used a semi-colon, but it is good practise to remember to insert it yourself.

To see some more commands, try

```
[  3+18;
[  40*21;
[  3!;
```

In the next example, you should only press the return/enter key after typing in the whole line:

```
[  2^3;exp(2);sin(PI);cos(3*PI/4);
```

The final example above illustrates some important information:

- Many commands can be put on a single line provided that each ends with its own colon/semi-colon.
- MuPAD is case-sensitive and `PI` in capitals represents the number 3.141592653... (more of this in a moment).

- Finally note that multiplication must be made explicit with the `*` symbol so that 3π is input as `3*PI`; powers are input via the `^` symbol as shown for 2^3 above.

You should also observe in the above examples that MuPAD uses exact arithmetic, giving exact answers wherever possible (so that $\sqrt{2}$ is not evaluated to a number, for example). If decimal approximations are wanted then in certain circumstances MuPAD can be forced to produce them by including at least one decimal point in the input, as illustrated in the following examples[1]:

```
[  sqrt(2.0);
[  22/7.0;
```

However, note that including a decimal point in the input does not guarantee a fully decimal output (try typing `3.0*Pi` at the prompt, for example). In Section 2.1.2 you will be introduced to the `float` command which *will* always return an output in decimal form.

Exercise 1.1
Use MuPAD to evaluate the following exactly:

$$19 \times 99, \quad 3^{20}, \quad 2^{1000}, \quad 25!, \quad \frac{3}{13} - \frac{26}{27}.$$

Now evaluate the following to ten significant figures:

$$\frac{21}{23}, \quad 17^{1/4}, \quad \frac{1}{99!}, \quad 0.216^{100}.$$

Note that the last two answers are so small that MuPAD gives them in 'floating-point' form.□

As you saw above with `PI`, MuPAD has some names reserved for constants. The list of these is short,[2] and the ones that you should be aware of now are shown in the Table below.

`PI`	$3.1415\ldots$
`I`	$\sqrt{-1}$
`infinity`	∞

Missing from this table, as there is no special symbol for it, is the number e, that is exp(1) or $2.7182\ldots$. You should not be tempted to input `e` or `E`; however, typing `exp(1)` at the prompt will give the required output

```
[  exp(1);
```

where the 'e' produced by MuPAD does indeed mean $2.7182\ldots$ (try typing `exp(1.0)` at the prompt to see what happens). Within MuPAD, you should be careful not to confuse the 'e' above, which when output denotes exp(1), with the letter e in italics which has no particular meaning.

Now work through the following exercise which uses the constant `I`.

Exercise 1.2
As stated above, MuPAD denotes the square root of -1 by `I`. Suppose that we wish to express $(2+3i)^4$ in the form $a+bi$, where a and b are real. The easiest way of achieving this is by

[1] These decimal numbers are given to ten significant figures, which is the default setting in MuPAD. You will see shortly how to alter this.

[2] type `?constants` at the prompt for further information.

typing

```
[  (2+3*I)^4;
```

Use MuPAD to express the following in the form $a + bi$:

$$(5 - 8i)^2,\ i^{-1},\ \frac{1+i}{5+2i} \quad \text{and} \quad \left(\frac{4}{3} + \frac{2i}{5}\right)^4.$$

□

1.3.2 Identifiers

Identifiers in MuPAD are names that may represent variables and unknowns. They are denoted by sequences of letters, digits and underscores (with the exception that an identifier may not begin with a digit). For example

```
x, f, a1, B2, initial_conditions
```

are all legitimate identifiers (remember that MuPAD is case-sensitive, so `a1` and `A1` are different identifiers). There are a few words that are reserved by MuPAD and so cannot be used as identifiers. You have already seen `PI`, which is one, and `for` and `while` are others, because these are needed for certain MuPAD operations (several of which you will learn about throughout this course).

Identifiers can be either *free* or *assigned*. A free identifier is one which has no value associated with it, whereas an assigned identifier takes a particular value (which might be a number, or a more complicated expression). To assign a value to an identifier use the assignment operator `:=`, that is, a colon followed by an equals sign. Hence `A := B` (followed by the usual colon or semi-colon and then the return/enter key) will assign to `A` the value of `B`. So typing

```
[  a:=20;
```

assigns the value 20 to the identifier `a`. This can be checked by asking MuPAD what value `a` takes, which is done simply by typing `a` followed by a semi-colon:

```
[  a;
```

Note that there is a direction implicit in assignment; typing `20:=a` does *not* assign the value `20` to the identifier `a`.

Extending this example, assign `b` the value `a+3` as follows:

```
[  b:=a+3;
```

MuPAD already knows that `a` has the value `20` and so has evaluated `a+3` directly as `23`.

Assignments can be over-written and MuPAD will only remember the most recent one that it has been asked to execute. So `b`, which above took the value `a+3`, can now be assigned the value `10` (say) as follows:

```
[  b:=10;
```

and check:

```
[  b;
```

Warning: *A frequent mistake in MuPAD is to attempt to use the equals sign* `=` *for assignment purposes, rather than the assignment operator* `:=`*. It is essential to use* `:=` *when assigning an*

expression to a name.[3]

The following illustration of assignment concerns the reserved identifier `DIGITS`[4], which controls the number of digits displayed as output and used in computation. The default value of `DIGITS` is 10 but this can be changed using the assignment operator, as shown here:

```
[  22/7.0;
[  DIGITS;
[  DIGITS:=20;
[  22/7.0;
```

To check your understanding of assigning variables, try the following exercises.

Exercise 1.3
Assign to `a` the value `10`, to `b` the value `20` and to `c` the value `a+2*b`. Then use MuPAD to evaluate `a+b+c`. □

Exercise 1.4
All rational numbers p/q, where p and q are integers ($q \neq 0$), have a terminating or (eventually) repeating decimal form. By increasing the number of `DIGITS` as appropriate, find the exact decimal form of 21/23. □

Exercise 1.5
Consider the following series of commands. After inputting them all as shown, what would the final line (that is `z;`) give as output? (Try to work this out *before* inputting the commands yourself!)

```
[  a:=3; b:=2;
[  // in the following line the output is deliberately suppressed
[  z:=a^2+b^2+x^2:
[  a:=4;
[  z;
```
□

Exercise 1.5 illustrates the use of one other new symbol, `//`, which is two forward slashes. MuPAD will ignore everything after this symbol, until the start of the next line. It helps to keep track of things if you put in lines of explanation like this from time to time; this will become particularly important when writing the longer procedures that will be encountered later in the course.

You may be wondering how to remove assignments to identifier names. The easiest way to do this is to type `reset():` or `reset();` (both have the same effect) at the prompt; this will reset everything[5] and enable you to start again from scratch. This is always an option if the expected output is not forthcoming; going back to the start of a sequence of command lines and adding in `reset():` before the first command might work wonders. Indeed, it is strongly recommended that you start all exercises with `reset():`. However, do bear in mind that `reset():` removes the effect of any previous output and so it should be used with caution. If you prefer to remove the assignment from one or more identifiers individually, then you can use `delete`:

[3] Uses of `=` in MuPAD will be illustrated later on in the course.
[4] The capitals are important here; `digits` and `Digits`, for example, are just free identifiers.
[5] `reset():` will reset `DIGITS` to 10, for example.

```
[  a:=3:  b:=4:  c:=5:
[  a; b; c;
[  delete a:
[  a; b; c;
[  delete b,c:
[  a; b; c;
```

1.3.3 Elementary functions

MuPAD has many in-built functions, most of which are fairly obvious. Several have already been introduced, for example:

```
[  cos(3*PI/4); sqrt(2.0); exp(5);
```

Note that, as you have already seen, where possible MuPAD will produce exact answers unless you request an output value in decimals by including a decimal point in the input.

For the moment, the functions that you should be familiar with are those in the table below.[6] (In each case the function needs to be followed by an argument in brackets, as illustrated in the examples above.)

`cos`, `cosh`	cosine and hyperbolic cosine
`sin`, `sinh`	sine and hyperbolic sine
`tan`, `tanh`	tangent and hyperbolic tangent
`sec`, `sech`	secant and hyperbolic secant
`csc`, `csch`	cosecant and hyperbolic cosecant
`cot`, `coth`	cotangent and hyperbolic cotangent
`exp`	exponential
`ln`	natural logarithm
`log(b,x)`	logarithm of x to base b
`abs`	absolute value
`sqrt`	square root

Before you practise these, here are some more examples.

```
[  (sin(PI/4))^2; sin(PI/4)^2; (sin(3))^2;
[  exp(2); ln(%);
[  log(10,1000);
[  abs(  -3);
```

In the above examples you should note the following:

[6] Remember that help with any of these commands may be found by typing `?` followed by the command; for example, `?abs` will give you more information on the `abs` command.

- MuPAD's notation for $\sin^2(x)$ (and indeed $\sin^3(x)$, $\cos^2(x)$, e.t.c.) is potentially misleading. When MuPAD outputs `sin(3)^2` in the example above, it means $\sin^2 3$, that is $(\sin 3)^2$, and not $\sin(3^2)$. Similarly, `sin(x)^2` or `(sin(x))^2` is used to input $\sin^2 x$; `sin^2x` would not be accepted as input.
- The symbol `%` represents 'the value of the expression most recently evaluated'. You should use it with great care but it can be helpful. It is better practice, however, to assign the output a name, and then to use this name in subsequent work, as illustrated by the following: `ans:=exp(2); ln(ans);`.
- Within reason, MuPAD is forgiving in allowing spaces, as illustrated in the final example above. However, it would not have evaluated `ab s(-3)`, for example.

To gain a little more practice with functions, try the following exercises.

Exercise 1.6
Use MuPAD to evaluate the following:

$$\tan(\pi/3),\ \cos^2(\pi/3),\ \sec(\pi/6),\ 1+\cot^2(\pi/4),\ \operatorname{cosec}^2(\pi/4).$$

Evaluate the following, giving numerical answers to ten significant figures where appropriate:

$$\sqrt{3+\sqrt{3+\sqrt{3}}},\quad e^{3\ln 4}.$$

□

Exercise 1.7
The binomial coefficient ${}^nC_r$ is evaluated in MuPAD using `binomial(n,r)`. Use MuPAD to evaluate ${}^{10}C_4$ and ${}^{250}C_{12}$. □

Exercise 1.8
Use the MuPAD help facilities to find out how to evaluate $\arcsin(1/2)$. Evaluate $\sec(\arctan x)$ in terms of x. □

Before finishing this section, we would like you to save your notebook, as described in Section 1.1.1. A filename such as Section1point3 would be sensible; remember that MuPAD will append the ending `.mn` to denote the fact that this is a MuPAD notebook. You will then need to open a new notebook (as described in Section 1.1.1) when you start Section 1.4. From now on, try to organize your MuPAD work into notebooks so that you can easily access the material you have covered as and when you need to. This will prove particularly helpful next term when you come to work through the projects.

That ends this first section on MuPAD. Before proceeding any further, you should take stock and make sure that you are happy with everything covered so far. In particular, you should feel confident using the material in the tables on pages 4 and 7 as well as the following: `:`, `;`, `:=`, `reset()`, `%`, `//` and `DIGITS`. The rest of this course will build on these commands and so it is essential that you understand them.

You may by now have come to the end of your first two hour practical session, in which case you should try to work through Section 1.4 in your own time (or, if you are unable to do this, you should resume your next scheduled session in the Statistics Department with Section 1.4). However, if you do still have some time left now then you should continue straight on. Section 1.4 is a brief introduction to MuPAD's powerful graphics facilities.

1.4 Simple 2D graphs with the `plot` command

One of the very useful features of MuPAD is the ease with which many different types of graphs may be drawn. This term we will introduce you to the basic commands and next term, if you wish, you may explore MuPAD's graphics packages further. Most graphs are drawn with variants of the `plot` command, which has several forms, each with many optional arguments that determine the appearance of the resulting graph. The simplest of these forms, with a few of the more important options, is introduced in this section; if at any time you wish to see a list of available options then type `?plot` at the prompt to access the relevant help pages.

1.4.1 Simple line graphs

The simplest graphs are those of functions of a single real variable; for example, the graph of the function $\sin(10/(1+x^2))$ for $0 \leq x \leq 10$ is given by the command[7]

```
[  plot(sin(10/(1+x^2)),x=0..10);
```

As usual, to save space, the output from the above command is not reproduced here. However, you should see the graph on your screen, and it should be blue, which is the default colour. The first argument in this command is the formula for the function and the second is the range[8] of x; if we needed the graph for $10 \leq x \leq 100$ then we would replace `x=0..10` by `x=10..100`. On the other hand, the command `plot(sin(10/(1+x^2)));` plots the graph over the default range $-5 \leq x \leq 5$.

The range of the vertical axis can also be specified; if we only wanted to see the values of $\sin(10/(1+x^2))$ in the range $0 \leq y \leq 1$ then we would use the command

```
[  plot(sin(10/(1+x^2)),x=0..10,ViewingBoxYRange=0..1);
```

Exercise 1.9
Assign the expression `sin(10*cos(x))` to the identifier `y` and use the command

```
plot(y,x=-PI..PI);
```

to draw the graph of $y = \sin(10\cos x)$ on the interval $[-\pi, \pi]$. □

The colour of a graph is changed using an optional argument of the plot command. For example, the graph of $y = \sin(10\sin x)$ for $0 \leq x \leq \pi$ is drawn in red with the command

```
[  plot(sin(10*sin(x)),x=0..PI,Colors=[RGB::Red]);
```

For a full list of colours that are predefined in MuPAD type `?Colors`[9] at the prompt.

One of the more useful features of the `plot` command is the ability to draw graphs of many functions on the same plot. Thus the graphs of $\sin x$ and $\cos x$ for $0 \leq x \leq \pi$ can be compared with the command

```
[  plot(sin(x),cos(x),x=0..PI,Colors=[RGB::Lavender,RGB::VenetianRed]);
```

Here the two functions are inserted one after the other, separated by a comma. Also we have used the `Colors` option in the form `Colors=[RGB::Lavender, RGB::VenetianRed]` where the square brackets are essential so that the colours are inserted as a list[10] and the colour order

[7] The '..' in `x=0..10` is input by typing two successive full stops.
[8] The word range here is a MuPAD term; it does not have the usual meaning of a the range of a function in mathematics.
[9] Note the use of American spelling here.
[10] You will meet lists in Section 3.6.2.

matches the function order. (It is not necessary to define colours when plotting two or more graphs, because MuPAD will automatically plot each curve in a different colour, but you may prefer to do so.) A legend can be added to the plot with the `LegendVisible` option as in the following example:

```
[  plot(sin(x),cos(x),x=0..PI,Colors=[RGB::Red,RGB::Blue],LegendVisible);
```

Any number of graphs, within reason, can be combined in this manner, simply by including more functions. Thus the command

```
[  plot(x,x^2,x^3,x=0..1,Colors=[RGB::Red,RGB::Blue,RGB::Black],LegendVisible);
```

draws the graphs of $y = x$ (red), $y = x^2$ (blue) and $y = x^3$ (black) for $0 \leq x \leq 1$ all on the same plot.

Exercise 1.10
Draw the graphs of the five functions

$y_1(x) = 1,\ y_2(x) = 1 + x,\ y_3(x) = 1 + x + x^2/2!,\ y_4(x) = 1 + x + x^2/2! + x^3/3!$ and $y_5(x) = \exp(x)$ on the same plot for $0 \leq x \leq 1$. □

If you place the cursor in the vicinity of the figure you have just drawn and click and hold the left mouse button, then you should see the coordinates of the cursor; this is helpful for finding approximate coordinates of roots or stationary points. To practise this, try the following exercise.

Exercise 1.11
Find the approximate coordinates of all the real solutions of the nonlinear simultaneous equations

$$\begin{aligned} y &= \sin x, \\ y &= x^3 - 5x^2 + 4. \end{aligned}$$

(You will see in Section 2.2.2 how to find numerical solutions to simultaneous equations such as these.) □

1.4.2 Parametric and polar plots

For a function that is defined parametrically (with x and y as functions of t, say) then the general syntax `plot([x(t),y(t)],t=t1..t2)` can be used to plot the graph of y against x for t from $t1$ to $t2$. So the ellipse $x = 2\cos t,\ y = \sin t$ is plotted with the command

```
[  plot([2*cos(t),sin(t)],t=0..2*PI);
```

Exercise 1.12
The cycloid

$$x = t - \sin t, \quad y = 1 - \cos t$$

is the curve traced out by a point on a wheel as the wheel turns. Plot this curve for $0 \leq t \leq 6\pi$. □

Polar plots are achieved as follows: the general syntax for plotting the graph of $r = f(t)$ for t from a to b is `plot(plot::Polar([f(t),t],t=a..b));`. This is illustrated by the following command which plots the cardioid $r = 1 - \cos\theta$:

```
[  plot(plot::Polar([1-cos(t),t],t=0..2*PI));
```

Exercise 1.13
Plot (both leaves of) the lemniscate $r^2 = \cos(2\theta)$. □

Exercise 1.13 completes this very brief introduction to MuPAD's graphics facilities. Once more you are encouraged to save your notebook for future reference. As mentioned earlier, you will have the opportunity to extend your knowledge of the various `plot` commands and options next term. In the meantime, in the next chapter you will look at how MuPAD may be used to evaluate, solve, differentiate and integrate expressions.

Chapter 2

Evaluating, solving and an introduction to calculus

In this chapter you will learn about:

- different ways of evaluating expressions;
- solving equations symbolically and numerically;
- using MuPAD to differentiate expressions;
- using MuPAD to evaluate integrals (symbolically and numerically).

2.1 Evaluating expressions

In this section `subs`, `|` and `float` are introduced. The material here is fairly brief; you will be given plenty of opportunity to practise the commands throughout the rest of the course. We start with `subs`.

2.1.1 Evaluating expressions at a point (or points)

The command `subs(object,Old=New)` replaces all occurrences of the expressions `Old` in `object` by the value `New`. For example, suppose we want to evaluate the expression $x^2 + 3x - 2$ at $x = 1$. This is achieved by using the commands

```
[  y:=x^2+3*x-2;
[  subs(y,x=1);
```

which obtains the required value. What is useful about this is that `y` itself has not changed, as verified by typing

```
[  y;
```

so that if we now wish to evaluate `y` at a different value of `x` then this is easily done.

Another method of achieving the same result is via the `|` operator (the notation here mimics the notation $f(x)|_{x=1}$):

```
[  x^2+3*x-2|x=1;
```

To obtain some practice on inputting expressions and evaluating them, try the following exercise.

Exercise 2.1
In each of the following, evaluate the expression at the given value.

(i) $x^3 - 3x^2 + 2x - 1$, at $x = 5$;

(ii) $\sin x \cos^3 x$, at $x = \pi/4$;

(iii) $\ln\left(u + 1 + \sqrt{u^2 - 3}\right)$, at $u = 2$. □

A single call of `subs` can be used to perform several substitutions. As an example, suppose that we wish to evaluate $r = x^2 + y^2 + z^2$ at the point $(x, y, z) = (1, 2, 3)$. This is achieved by typing

```
[  subs(x^2+y^2+z^2,x=1,y=2,z=3);
```

to give $r = 14$.

Now we move on to `float` which evaluates a quantity numerically whenever this is possible.

2.1.2 `float`

The `float` command is used to evaluate an expression to a *floating-point* number, that is, a number rounded to a certain number of significant figures. The default number of significant figures used is 10 (the default value of `DIGITS`). This may be changed by re-assigning `DIGITS`[1] – which will then produce a global change to all subsequent expressions until it is re-assigned again:

```
[  float(sqrt(2));
[  DIGITS:=20:  float(sqrt(2));
[  DIGITS:=10:
```

Exercise 2.2
The following two expressions are both approximations to π that were discovered by the Indian mathematician Ramanujan (1887–1920):

$$\pi_1 = \frac{12}{\sqrt{190}} \ln((2\sqrt{2} + \sqrt{10})(3 + \sqrt{10}))$$

and

$$\pi_2 = \sqrt{\sqrt{9^2 + \frac{19^2}{22}}}.$$

Use the `float` command to find the absolute errors $|\pi_1 - \pi|$ and $|\pi_2 - \pi|$. Hence determine how good these approximations actually are. □

That completes this brief introduction to evaluation. We now move on to look at solving equations.

2.2 The `solve` command

The main aim of this section is to introduce `solve`, which can be used to solve equations, inequalities, systems of these, differential equations, recurrence relations, etc. In this section

[1] You saw a re-assignment of `DIGITS` in Section 1.3.2.

we only look at the solution of equations, both analytically and numerically. You will also learn about the `assign` command.

2.2.1 Solving equations symbolically

The `solve` command can be used to rearrange simple algebraic expressions to arrive at a new expression. For example, the solution of the equation

$$2x + 3 = 0 \tag{2.1}$$

for x can be obtained using MuPAD as follows:

```
[  eq:=2*x+3=0;
[  solve(eq,x);
```

Note the different uses of `:=` and `=` here; `eq` is assigned the MuPAD version of Equation (2.1). The command `solve(eq,x)` has solved the equation assigned to `eq` for the variable x. In general, then, the syntax for solving one or more equations is `solve(S,X)` where `S` is either a single equation or a set of equations and `X` is the required variable or set of variables. So suppose that we wish to solve the simultaneous equations

$$x + y = 2, \quad -x + 3y = 3.$$

This is done within MuPAD as follows:

```
[  delete x,y:
[  eq:={x+y=2,-x+3*y=3}:
[  sol:=solve(eq,{x,y});
```

(The curly brackets in the input here are essential and denote a set; as mentioned earlier sets will be discussed in more detail in Section 3.6.1.) A (rather clumsy) verification that this is correct can be obtained by evaluating `eq` at the solution point, using `|`:

```
[  eq|sol;
```

Note that at this stage the variables x and y have not actually been assigned the values contained within the solution `sol`. This is confirmed by typing

```
[  x,y;
```

to see that x and y are returned unassigned. If we wished to assign x and y the values given in the solution then we could do so by typing

```
[  x:=3/4:  y:=5/4 :
[  x,y;
```

However, this is clearly long-winded and error prone; a neater way is to use the `assign` command as follows:

```
[  delete x,y:
[  eq:={x+y=2,-x+3*y=3}:
[  sol:=solve(eq,{x,y});
[  assign(sol[1]):
[  x,y;
```

To understand what has happened here, type the following:

```
[  delete a:
[  assign({a=3});
[  a+2;
```

In this simple example, `a` has been assigned the value `3`, the effect of which is the same as if we had typed `a:=3:`.

In general, `assign({A=B})` can be thought of as having the same effect as `A:=B`, with one big difference. Once `A` has been assigned a value then `assign` as illustrated above cannot be used on `A` again; `A` has to be unassigned first.

To restore `a` to its unassigned status use the `delete` command:

```
[  delete a:
```

The following exercises give you some practice in the use of `solve` and `assign`.

Exercise 2.3
Find the solutions of the following equations:

(a) $x^2 - x - 2025 = 0$, (b) $x^3 - 6x^2 - 19x + 24 = 0$,

(c) $2x^4 - 11x^3 - 20x^2 + 113x + 60 = 0$.

(Please be sure to enter the expressions in (b) and (c) carefully; MuPAD needs to be able to factorize the polynomials and may not be able to do so if you make a mistake when typing them in.) □

Exercise 2.4
Use the `solve` command to find the point of intersection, in the (x, y)- plane, of the two lines

$$ax + by = A, \quad cx + dy = B.$$

Using `assign`, find an expression for the distance of the point of intersection from the origin, namely $\sqrt{x^2 + y^2}$.

2.2.2 Solving equations numerically

The `solve` command is used for finding *symbolic* solutions to equations. This may not always be possible, but nevertheless *numerical* solutions can usually be found. For example, suppose we wish to solve the equation

$$\sin x = x^3 - 5x^2 + 4,$$

and that having plotted the graphs of $\sin x$ and $x^3 - 5x^2 + 4$ we know that there are three solutions, at approximately $x = -0.90,\ 0.89$ and 4.78 (these approximate solutions were obtained in Exercise 1.11). To find the negative solution more precisely, one way is to type

```
[  float(solve(sin(x)=x^3-5*x^2+4,x=-1..0));
```

The negative solution is $x = -0.900\,400\,208\,9$ to ten significant figures.[2] Note that the range is specified using the `x=-1..0` construction; although this is optional it is always advisable to assist MuPAD by giving it a range in which to search for solutions. If not, it might find a different solution to the one you want. The other two solutions of $\sin x = x^3 - 5x^2 + 4$ can be found by specifying different ranges:

[2]The solution is given to ten significant figures because that is the current value of `DIGITS`.

```
[  float(solve(sin(x)=x^3-5*x^2+4,x=0..1));
[  float(solve(sin(x)=x^3-5*x^2+4,x=4..5));
```

However, as you may have realised, with this syntax MuPAD first tries to solve the equation symbolically and this can take time. To avoid the symbolic preprocessing, the `numeric::solve()` structure can be used as follows:

```
[  numeric::solve(sin(x)=x^3-5*x^2+4,x=4..5);
```
which you will see is far quicker!

To test your understanding, use the `numeric::solve()` structure in the following exercises.

Exercise 2.5
Find real solutions of the equation

$$x \sin x = \frac{1}{2}$$

for x in the following ranges: (i) $0 < x < 2$; (ii) $2 < x < 4$; (iii) $6 < x < 7$; (iv) $8 < x < 10$. Verify graphically that there are no solutions in the range $4 < x < 6$. □

Exercise 2.6
Use `plot` to find the number and approximate value(s) of the real solution(s) of $x^3 + 3x^2 - 2x + 1 = 0$. Then use `numeric::solve` to evaluate all the solutions of $x^3 + 3x^2 - 2x + 1 = 0$.□

That completes the work of this section, which was primarily to introduce the command `solve`. By itself, it can be used if exact symbolic solutions are required, whereas `numeric::solve` is very useful in obtaining approximate numerical solutions to equations. There will be more opportunities to practise these commands in the remainder of this course. For now we move on to look at how MuPAD can be used to solve problems in calculus, starting with differentiation.

2.3 Differentiation and the `diff` command

MuPAD can easily find the derivatives of expressions of one or several variables. In this section we introduce the `diff` command, which is the basic differentiation operator for expressions, and illustrate its use in different situations including finding the coordinates of stationary points.

The use of `diff` is best illustrated with some simple examples. The derivative of $\sin x$ with respect to x is found with the command

```
[  diff(sin(x),x);
```

The general syntax for differentiating an expression `expr` with respect to a variable `var` is

```
[  diff(cos(y),y);
```

and the derivative of $e^{x^2+a^2}$ with respect to x is found with the commands

```
[  z:=exp(x^2+a^2):
[  d1:=diff(z,x);
```

The second derivative of $e^{x^2+a^2}$ with respect to x is

```
[  diff(d1,x);
```

and this same result can also be found all in one go by typing

```
[  diff(z,x,x);
```

or the equivalent

```
[  diff(z,x$2);
```

where `x$2` is short for `x,x` (try typing just `x$2` at the prompt).

So the fifth derivative of $\sin(x^2)$ is achieved by typing `diff(sin(x^2),x,x,x,x,x);` or with the command

```
[  diff(sin(x^2),x$5);
```

Partial differentiation[3] of expressions of more than one variable is performed by a straightforward extension of this procedure. For example, consider the expression $\sin x \cos y$; then its second derivatives are found as follows. First define $z = \sin x \cos y$:

```
[  z:=sin(x)*cos(y);
```

Then the second derivative $\partial^2 z/\partial x^2$ is given by

```
[  diff(z,x$2);
```

whilst $\partial^2 z/\partial y^2$ is found with

```
[  diff(z,y$2);
```

For the mixed second derivative $\partial^2 z/\partial x \partial y$ type

```
[  diff(z,x,y);
```

and to verify that the order of differentiation does not matter:

```
[  diff(z,y,x);
```

If we wanted $\partial^5 z/\partial x^2 \partial y^3$, then we would type

```
[  diff(z,x$2,y$3);
```

Sometimes it is necessary to find the derivative of an expression at a particular point, rather than as a function of the independent variable(s). This is achieved with a combination of the `float`, `subs` and `diff` commands[4]. Thus the value of the first derivative of $\sin x$ at $x = 1.6$ is given by the single command

```
[  float(subs(diff(sin(x),x),x=1.6));
```

The following exercises give practice in the `diff` command and also revise some earlier commands such as `solve` and `plot`.

Exercise 2.7
Find

$$\frac{d}{dx}\left(\frac{(x^2+1)^4}{e^{2x}}\right)$$

and evaluate it at $x = 1$. □

Exercise 2.8
Plot the second derivative of $x^4/(1+x^2)$ for $-5 \leq x \leq 5$. □

Exercise 2.9
Given $y = 12x^5 - 15x^4 + 20x^3 - 330x^2 + 600x + 2$, find the (x, y) coordinates of any stationary points (that is, those for which $y'(x) = 0$). Use MuPAD to plot y over a range of x large enough to include all stationary points. □

[3] This topic may be new to you; it will be covered in lectures this term.

[4] `subs` and `float` were introduced in Section 2.1

Exercise 2.10
If $z = \ln(x^2 + y^2)$ find the value of the constant a such that

$$\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 = ae^{-z}.$$

□

2.3.1 Differentiation of arbitrary functions

So far only the derivatives of expressions with an explicit form (in terms of one or more independent variables) have been found. In this section we look at how to differentiate arbitrary functions, say $y(x)$ or $f(x, y)$, without first giving them an explicit form. For example, we might want to find dy/dx (as a function of y and x) given that $x^2 + y^2 = 3$. The problem is that typing

```
[  diff(y,x);
```

returns `0` (rather than the `dy/dx` that might have been expected) because MuPAD does not know that `y` depends on `x`; `x` and `y` are free independent variables and are treated as such.

This difficulty has been addressed in MuPAD by allowing the use of `y(x)`; MuPAD interprets this syntax to mean that, for the purposes of differentiation and integration, `y(x)` depends on `x`. Thus we have

```
[  diff(y(x),x);
```

```
[  diff(y(x),x$2);
```

and indeed

```
[  diff(z(p),p);
```

(where `z` has been unassigned first if necessary) and even

```
[ diff(z(x,y),x,y$2);
```

MuPAD interprets the syntax `y(x)` (or its equivalent in terms of other letters) in a manner consistent with the rules of calculus. With this notation the `diff` command knows all the usual rules of differentiation, for instance the addition, multiplication and quotient rules:

```
[  diff(f(x)+g(x),x);
```

```
[  diff(f(x)*g(x),x);
```

```
[  diff(f(x)/g(x),x);
```

To illustrate an application of this theory, suppose that we wish to find dy/dx (in terms of x and y) given that $x^2 + y^2 = 3$. To do this with MuPAD, first define the given explicit equation by typing

```
[  eq1:=x^2+y(x)^2=3;
```

and then differentiate this equation, assigning the output equation to a new name, `eq2`:

```
[  eq2:=diff(eq1,x);
```

Finally rearrange this new equation to make dy/dx the subject:

```
[  solve(eq2,diff(y(x),x));
```

Hence (if we assume $y(x) \neq 0$) $dy/dx = -x/y$.

Now try the following exercises.

Exercise 2.11
Find the derivatives of

$$\sin f(x), \quad \sin\left(e^{f(x)}\right), \quad \exp\left(\sqrt{1+f(x)g(x)}\right).$$

□

Exercise 2.12
Use the `solve` command to find dy/dx (in terms of y and x) given that

$$x^2 + y^2 + 3xy = 0.$$

□

That concludes the work on the differential operator `diff`. In the next section we stay with calculus and look at integration.

2.4 Integration

MuPAD can evaluate many integrals symbolically, and it can also provide numerical estimates of most definite integrals which cannot be evaluated symbolically as well as those that can. In this section the integration command `int` is introduced.

2.4.1 The `int` operator

The syntax for integrating an expression `expr` with respect to a variable `var` is `int(expr,var)`, and this is best explained with a simple example. To integrate the expression xe^{ax^2} with respect to x type

```
[  int(x*exp(a*x^2),x);
```

Note that MuPAD does not include the arbitrary constant of integration.

Definite integrals are input using the `x=a..b` construction which you have seen before, in `plot` and `numeric::solve`, for example. So to evaluate the definite integral

$$\int_0^1 xe^{5x^2}\,dx$$

type

```
[  int(x*exp(5*x^2),x=0..1);
```

and to evaluate

$$\int_0^u \frac{dx}{\sqrt{u-x}}$$

type

```
[  int(1/sqrt(u-x),x=0..u);
```

(in the `x=a..b` construction `a` and `b` can be variables, numbers, or expressions).

If MuPAD cannot evaluate an integral then it simply returns it:

```
[  int(x^x,x);
```

At this point you really need to experiment with some integrals just to see what can be done.

Exercise 2.13
Use MuPAD to integrate the following expressions with respect to x:

(i) $\sqrt{e^x - 1}$, (ii) $x^2(ax+b)^{5/2}$, (iii) $\sinh(6x)\sinh^4(x)$, (iv) $\cosh^{-6}(x)$, (v) $\sin(\ln(x))$,

where a and b are constants. □

Exercise 2.14
Use MuPAD to evaluate the following expressions symbolically:

(i) $\displaystyle\int_{1/2}^{1} \frac{1}{1+x^3} dx$, (ii) $\displaystyle\int_0^1 x^2 \tan^{-1} x \; dx$, (iii) $\displaystyle\int_0^\infty \frac{1}{(1+x)(1+x^2)} \; dx$. □

From the exercises you have just done you will observe that MuPAD's ability to do rather nasty integrals is impressive, and that it easily evaluates all of the integrals found in elementary texts as well as most of the indefinite integrals given in standard tables.

Nevertheless one sometimes encounters cases where it is desirable to have MuPAD evaluate an integral numerically, and this is discussed in the next section.

2.4.2 Numerical evaluation of integrals

You may be aware that the majority of integrals cannot be evaluated symbolically. If MuPAD fails to find a symbolic answer to a definite integral, or if you know that the integral in question cannot be evaluated in terms of known functions, then it may be necessary to evaluate it numerically. This can be done using the `float(int())` construction, but (as with `solve`) MuPAD will try to evaluate the integral symbolically first, and this can be time-consuming. To compute numerically from the beginning, use `numeric::solve`, as illustrated in the following example, where we try to evaluate

$$\int_0^1 e^{-t} \arcsin(t) \; dt.$$

First attempt to find a symbolic result by typing

```
[  z:=int(exp(-t)*arcsin(t),t=0..1);
```

However, MuPAD just returns the integral, which means that it does not know the answer. To find the numerical value of the integral simply evaluate `z` to a floating point number:

```
[  float(z);
```

Alternatively, to do the whole thing in one go then type

```
[  numeric::int(exp(-t)*arcsin(t),t=0..1);
```

Note that the use of `numeric::int` here saves MuPAD the time-consuming futile attempt at evaluating the integral symbolically before finding the numerical solution.

Exercise 2.15
Use MuPAD to evaluate the following integrals numerically to six digit accuracy:

(i) $\displaystyle\int_0^1 e^{x^3} dx$, (ii) $\displaystyle\int_0^{10} \frac{1}{\sqrt{1+x^4}} \; dx$, (iii) $\displaystyle\int_0^5 \sin\left(e^{x/2}\right) dx$. □

That concludes this section on integration; you are now half way through the course. If you find that you have time to spare in your laboratory session then you might like to work through Appendix A. Or try the following optional extended exercises.

Exercise 2.16
In this optional exercise you are going to explore the function $\operatorname{sinc} x$ which is defined as

$$\operatorname{sinc} x = \begin{cases} \dfrac{\sin x}{x} & x \neq 0, \\ 1 & x = 0. \end{cases}$$

(a) Assign the expression `sin(x)/x` to the variable `f` in MuPAD and then plot `f` for $-30 \leq x \leq 30$. (Note that MuPAD has no problem with plotting $\frac{\sin 0}{0}$. However, you should satisfy yourself that your graph looks correct at $x = 0$, given the above definition.)

There are many interesting and unusual properties of the sinc function and you are asked to investigate two of these properties here.

(b) First, it can be shown that each maximum or minimum of the graph of $\operatorname{sinc} x$ corresponds to a point of intersection of the graphs of $\operatorname{sinc} x$ and $\cos x$. Use MuPAD to illustrate this, by drawing the graphs of $\operatorname{sinc} x$ and $\cos x$ on the same plot for $-10 \leq x \leq 10$. Then use `numeric::solve` to find numerical approximations for the x-coordinates of all stationary points of $\operatorname{sinc} x$ for $0 < x \leq 10$, and verify that these are indeed where the two graphs that you have just drawn meet.

(c) Use MuPAD to evaluate the three integrals

$$\text{(i)} \int_0^\infty \operatorname{sinc} x \; dx, \quad \text{(ii)} \int_0^\infty \operatorname{sinc} x \operatorname{sinc}\frac{x}{3} \; dx, \quad \text{(iii)} \int_0^\infty \operatorname{sinc} x \operatorname{sinc}\frac{x}{3} \operatorname{sinc}\frac{x}{5} \; dx.$$

What do you think the value of $\displaystyle\int_0^\infty \prod_{k=0}^{5} \operatorname{sinc}(x/(2k+1)) \; dx$ is? Use MuPAD to verify your conjecture. □

Exercise 2.17
In this optional exercise you are asked to use MuPAD to investigate some rational bounds on π.

(a) Verify that

$$\int_0^1 \frac{x^4(1-x)^4}{1+x^2} dx = \frac{22}{7} - \pi. \tag{2.2}$$

This is in itself an interesting result, as $\frac{22}{7}$ is often used as a rational approximation for π. However, this result can also be used to obtain bounds on π in the form $a/b < \pi < c/d$, where a, b, c and d are integers.

(b) Verify (without using MuPAD) that

$$\frac{1}{2}\int_0^1 x^4(1-x)^4 dx < \int_0^1 \frac{x^4(1-x)^4}{1+x^2} dx < \int_0^1 x^4(1-x)^4 dx.$$

(c) Evaluate $J = \displaystyle\int_0^1 x^4(1-x)^4 dx$, and hence deduce that

$$\frac{1979}{630} < \pi < \frac{3959}{1260}. \tag{2.3}$$

This gives rational bounds on π. You can now use MuPAD to obtain tighter rational bounds, in the following manner.

It can be shown that the identity (2.2) can be generalized by replacing the powers of 4 by powers of any integer multiple of 4 in the following manner:

$$\int_0^1 \frac{x^{4n}(1-x)^{4n}}{2^{2(n-1)}(1+x^2)}dx = (-1)^n(\pi - R_n),$$

where $n = 1, 2, 3, \ldots$ and R_n is a rational number. (In (2.2) above, $n = 1$ and $R_1 = 22/7$.) MuPAD can verify this for any given n and the corresponding value of R_n can be computed. Then, following the procedure of parts (ii) and (iii) above, new bounds on π can be found[5].

(d) Use MuPAD to calculate R_5.

(e) Hence find the upper and lower bounds on π for the case when $n = 5$. Evaluate these both as rational numbers and in decimal form, to thirty significant figures. □

[5]It can be shown that the higher the value of n, the tighter the bounds on π, but this is not considered here.

Chapter 3

More MuPAD commands

In this chapter you will learn about:

- solving differential equations and recurrence relations;
- sums and limits in MuPAD;
- defining functions in MuPAD;
- the use of sets and lists in MuPAD.

3.1 Solving ordinary differential equations using `solve`

The `solve` command, which you studied in Section 2.2.1, can be used with `ode` to find the closed form solutions of many types of differential equations. For example, to solve the homogeneous differential equation

$$\frac{d^2y}{dx^2} - y = 0$$

type

```
[  solve(ode(diff(y(x),x$2)-y(x)=0,y(x)));
```

Notice the slightly strange way by which MuPAD gives the constants of integration; the solution is $y = Ae^x + Be^{-x}$ where A and B are constants. The important thing here is that `y(x)` is input and not just `y`; this is an example of MuPAD differentiating arbitrary functions which was discussed in Section 2.3.1.

Inhomogeneous equations are no different; to solve

$$\frac{dy}{dx} + \frac{y}{x} = x^2$$

type

```
[  solve(ode(diff(y(x),x)+y(x)/x=x^2,y(x)));
```

and the answer is seen to be $y = x^3/4 + c/x$ for some constant c.

MuPAD does not suffer from fatigue when the going gets tough – although it may need some extra help – the solution of

$$\frac{d^3x}{dt^3} - 3\frac{d^2x}{dt^2} + 3\frac{dx}{dt} - x = 16e^{3t}$$

is found by typing

```
[  solve(ode(diff(x(t),t$3)-3*diff(x(t),t$2)+3*diff(x(t),t)-x(t)=16*exp(3*t),x(t)));
[  simplify(%);
```

(The use of the command `simplify` is explained in Section B.3.) To insert initial and/or boundary conditions into MuPAD, proceed as in the following examples. The solution of the boundary value problem

$$y'' + 5y' + 6y = 0, \quad y(0) = 0, \quad y(1) = 3$$

is found with the following commands. First define the differential equation by typing

```
[  de:=diff(y(x),x,x)+5*diff(y(x),x)+6*y(x)=0;
```

and then solve it with

```
[  solve(ode({de,y(0)=0,y(1)=3},y(x)));
```

Note how the differential equation and boundary conditions are separated by commas and enclosed in curly brackets.

Now consider the initial value problem

$$y'' = y, \quad y(0) = 1, \quad y'(0) = 0.$$

It is solved with the following command:

```
[  solve(ode({diff(y(x),x,x)=y(x),y(0)=1,y'(0)=0},y(x)));
```

Exercise 3.1
Use MuPAD to solve the following differential equations:

(i) $\dfrac{d^2y}{dx^2} + 3\dfrac{dy}{dx} + 4y = \sin x,$

(ii) $\dfrac{dy}{dx} = 3xy, \ y(0) = 1.$

□

Exercise 3.2
Solve the differential equation $y'' + y = 0$, subject to the initial conditions $y(0) = a, \ y'(0) = b$. Assign the result to a variable and then evaluate the solution at $x = \pi$. □

3.2 Solving recurrence relations with `solve`

Recurrence relations are easy to solve using MuPAD's `solve` command with `rec`, which works in much the same way as `solve` with `ode` above. Consider the sequence defined by

$$x_0 = 1, \quad x_{n+1} = n + 1 + x_n, \quad n = 0, 1, 2, \ldots.$$

To find the n^{th} term x_n type

```
[  xn:=solve(rec(x(n+1)=n+1+x(n),x(n),{x(0)=1}));
```

Note that in this case it is only the initial condition that is placed in curly brackets; strictly speaking this is not necessary if there is only one condition, but if there were two (or more) then the brackets would be needed.

Exercise 3.3
The Fibonacci numbers form a sequence defined by the recurrence relation

$$u_0 = 0, \quad u_1 = 1, \quad u_n = u_{n-1} + u_{n-2}, \quad n = 2, 3, 4, \ldots.$$

Use MuPAD to find an expression for u_n and hence verify that $u_{20} = 6765$. □

3.3 Sums and products

In this section the `sum` and `product` commands are introduced. We will concentrate on `sum`, leaving `product` for an exercise at the end of the section.

Suppose that it is required to evaluate the sum S defined by

$$S = \sum_{r=1}^{10} r = 1 + 2 + 3 + \cdots + 9 + 10 = \frac{1}{2} \times 10 \times 11 = 55.$$

To find this using the `sum` command type

```
[  S:=sum(r,r=1..10);
```

The syntax for the `sum` command is `sum(expr,r=a..b)`, where the first argument `expr` is the expression to be summed and the second argument `r=a..b` shows the variable `r` running from `a` to `b` (inclusive, incremented by 1). You have already seen the `r=a..b` construction, for example in the `plot` and `int` commands, although when used with `sum` the limits `a` and `b` must be integers and the increment is always one.

Let us now consider a slightly more general case, where the upper limit is replaced by a free variable, n. Then

$$S = \sum_{r=1}^{n} r = 1 + 2 + 3 + \cdots + (n-1) + n = \frac{1}{2} n(n+1). \tag{3.1}$$

and the `sum` command yields the standard result:

```
[  S:=sum(r,r=1..n);
```

Infinite sums are evaluated using `sum` and `infinity`; for example, the infinite geometric series is found by typing

```
[  sum(a^k,k=1..infinity);
```

Now try the following exercises.

Exercise 3.4
Use MuPAD to evaluate the following:

(i) $\displaystyle\sum_{r=1}^{10} (r^2 + 3r - 2)$, (ii) $\displaystyle\sum_{i=1}^{50} 2^i$, (iii) $\displaystyle\sum_{k=0}^{n-1} a^k$, (iv) $\displaystyle\sum_{k=0}^{\infty} \frac{1}{k!}$. □

Exercise 3.5
Use MuPAD to verify that

(i) $\displaystyle\sum_{n=1}^{\infty} \frac{2}{(n+1)(n+2)} = 1$, (ii) $\displaystyle\sum_{k=0}^{\infty} \frac{k^2 + k - 1}{(k+2)!} = 0$, (iii) $\displaystyle\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$. □

The `float` command (introduced in Section 2.1.2) can be used with `sum` to evaluate finite sums numerically. For example, the sum

$$\sum_{k=1}^{100} \frac{1}{k+k^{3/2}}$$

can be evaluated to ten significant figures with the command

```
[  float(sum(1/(k+k^(3/2)),k=1..100));
```

Now try the following exercises.

Exercise 3.6
Find the numerical values of the sums

(i) $\sum_{k=1}^{10} e^{-\sqrt{k}}$, (ii) $\sum_{k=1}^{100} \frac{1}{\sqrt{k}}$. □

Exercise 3.7
By embedding one `sum` command within another, evaluate the double sums

$$\text{(i)} \sum_{m=0}^{10}\sum_{n=0}^{10}(2n+1)e^m, \quad \text{(ii)} \sum_{n=-20}^{20}\sum_{m=-10}^{10} \frac{1}{n^2+\ln(22+m)}.$$

□

That almost concludes this section on the `sum` command. However, you may be wondering if there is a command similar to `sum`, but for evaluating products. MuPAD does indeed have `product`, and this is explored in this final exercise:

Exercise 3.8
Assuming that `product`[1] behaves in the same way as `sum`, use MuPAD to evaluate the following:

(i) $\prod_{r=1}^{10} r^2$, (ii) $\prod_{r=1}^{5}(1-q^r)$.

Note: If you have not seen it before, $\prod$ denotes a product in the same way as $\sum$ denotes a sum. For example,

$$\prod_{r=1}^{5}(r+2) = 3\times 4\times 5\times 6\times 7 = 2520.$$

□

3.4 Using MuPAD to evaluate limits

This short section introduces the command `limit`. Its syntax is fairly self-explanatory; for example to find

$$\lim_{x\to 0} \frac{\sin x}{x}$$

type

```
[  limit(sin(x)/x,x=0);
```

[1] Remember that you can obtain more information by typing `?product` at the prompt.

Left and right limits are also found easily; for example to find

$$\lim_{x\to 3^+} \frac{x^2-4}{x^2-5x+6}$$

use the command

```
[  limit((x^2-4)/(x^2-5*x+6),x=3,Right);
```

Exercise 3.9
Use MuPAD to evaluate the following limits

(i) $\lim_{x\to 0} \dfrac{\tan x - x}{x - \sin x}$, (ii) $\lim_{x\to\infty} \left(1+\dfrac{1}{x}\right)^x$. □

3.5 Functions in MuPAD

Up to now we have not defined a function in MuPAD; we have just used 'expressions' such as `y:=x^2+3`. Here `y` is a variable which has been assigned the value `x^2+3`, and we cannot evaluate $y(2)$ by typing `y(2);`.

Functions (in the mathematical sense) are defined using the `->` construction, that is a minus sign followed by the greater than symbol. This is best illustrated with an example. Suppose we wanted to define the function f such that $f(x) = x^2 \sin x$. In MuPAD this is achieved by typing

```
[  f:=x->x^2*sin(x);
```

From this things like $f(\pi/2)$ and $f(2x)$ can be evaluated, and $df(x)/dx$ and $df(t)/dt$ can be found:

```
[  f(PI/2);
[  f(2*x);
[  diff(f(x),x);
[  diff(f(t),t);
```

Note that `diff` always returns an expression (rather than a function, for example). There is a way around this, using the operator `D` instead of `diff`, but we will not go into details here.

Functions of more than one variable are defined in much the same way, with the variables enclosed in left and right parentheses as illustrated for the function g defined by $g(s,t) = e^{is}(t^2 - t + 1)$:

```
[  g:=(s,t)->exp(I*s)*(t^2-t+1);
[  g(PI,1);
```

Compositions of functions are achieved with the 'at' symbol @, with `(f@g)(x)` representing the composition $f(g(x))$. This is illustrated with the following commands:

```
[  f:=x->x^2-x;
[  g:=x->x^3;
[  (f@g)(x);
[  (f@f)(t);
```

```
[  expand(%);
```

(The use of the command `expand` is explained in Section B.1.)

Exercise 3.10
If f is defined by $x \to x^3 \ln(x^2)$ and g is defined by $x \to 2x^3$, what are $f'(x)$ and $g'(x)$? Obtain an expression for $g'(f'(x))$. □

Exercise 3.11
For f and g as defined in Exercise 3.10, verify the Chain Rule for the derivative of $f(g(x))$, ie that if $h(x) = f(g(x))$ then $h'(x) = f'(g(x))g'(x)$.
Hint: You might find it easier to find expressions for $h'(x)$ and $f'(g(x))g'(x)$ separately; then subtract one from the other and you should obtain zero, although you may need to use `simplify` as described in Section B.3 to see this. □

3.6 Sets and lists in MuPAD

In this final section of Chapter 3 we look briefly at how MuPAD deals with sets and lists.

3.6.1 Sets

A set in MuPAD takes the form {`expr1, expr2, expr3,..., exprn`}, in other words it is a sequence of expressions enclosed in curly brackets. Examples of sets are

```
[  S1:={1,2,3};
```

```
[  S2:={x^2,3*x,2};
```

Note that these are sets in the mathematical sense of the word; order is not important and repetitions are ignored:

```
[  1,3,4,1,2,1;
```

Given two sets A and B, MuPAD can perform the usual set operations of union $A \cup B$, intersection $A \cap B$ and difference $A \setminus B$. These are illustrated in the following examples.

```
[  A:={1,2,3,4};
```

```
[  B:={2,4,6,8};
```

```
[  A union B;
```

```
[  A intersect B;
```

```
[  A minus B;
```

The jth element of the set S is `S[j]`:

```
[  B[3];
```

You have in fact already seen examples of sets in this manual. For instance, on p. 14 the commands

```
[  eq:={x+y=2,-x+3*y=3}:
```

```
[  sol:=solve(eq,{x,y});
```

that solved a pair of simultaneous equations used sets for both the input equations and the output variables.

Exercise 3.12
Use MuPAD to verify that if `X:={1,2,3,5}`, `Y:={2,3,4}` and `Z:={1,3}` then

$$X - (Y \cup Z) = (X - Y) \cap (X - Z) \quad \text{and} \quad X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z).$$

(Note that MuPAD will never be able to *prove* the generality of these results.) □

3.6.2 Lists

In MuPAD, a list takes the form `[expr1, expr2, expr3,..., exprn]`, in other words it is a sequence of expressions enclosed in square brackets. Here order and repetition are both important. Examples of lists are

```
[  L1:=[1,2,3,4];
```

```
[  L2:=[1,2,1,3,2];
```

As with sets a specific element can be extracted:

```
[  L2[3];
```

and lists can be concatenated using the `op` command:

```
[  L3:=[op(L1),op(L2)];
```

We can also find out how many elements are in a list:

```
[  nops(L3);
```

You will revisit lists in the next chapter when you look at procedures.

That completes this third chapter of the manual. It has been kept deliberately short so that if you have fallen behind you have a chance to catch up. Alternatively, if you have time to spare within your two hour session then you might like to work through some of the material in Appendices A and B. Or you could proceed directly to Chapter 4.

Chapter 4

Loops, conditionals and procedures

In this chapter you will learn about:

- using MuPAD loops and conditionals;
- procedures in MuPAD;
- debugging MuPAD programs.

In this chapter you will consolidate all you have learned so far and you will also be introduced to some simple programming techniques. The material here is very important and a lot of it will be needed in the Hilary Term projects; it is therefore crucial that you start Hilary Term Week 3 having completed Chapter 4.

As stated at the start of this manual, as well as being a highly effective interactive symbolic calculator, MuPAD is also a programming language. By this we mean that, in MuPAD, a series of command lines can be written which, when executed, perform a particular task. You have seen several simple programs already (and hopefully written a few too). In this chapter you will look at programming in more detail, to develop longer and more structured command sequences. In particular you will look at some of the main building blocks of programming, namely loops, conditionals and procedures.

4.1 Loops

Sometimes a MuPAD command (or a sequence of commands) may need to be executed a number of times. The `for...do...end do;` control structure is provided exactly for this purpose. For example, say we wanted to output the squares of the integers from 1^2 to 5^2. This is achieved with the following command:

```
[  for j from 1 to 5 do print(j^2) end_for;
```

where `end_for` is `end` 'underscore' `for`. The general structure is

`for` *<var>* `from` *<expr>* `to` *<expr>* `do` *<statement sequence>* `end_for;`

where *var* is a variable (`j` in the example above), *expr* are expressions (`1` and `5` above) and *statement sequence* is the expression we want executed (`j^2` above).

Note that the `print` command is used above to generate the result for each `j`; compare with

```
[  for j from 1 to 5 do j^2 end_for;
```

By default `j` was incremented by 1 in the above example; a different step size is achieved with the optional `step`:

```
[  for j from 1 to 9 step 2 do print(j^2) end_for;
```

and the general structure now is

```
for <var> from <expr> to <expr> step <expr> do <statement sequence> end_for;
```

As another example, this construction can be used (instead of `sum`) to find the sum of the cubes of the even integers between 50 and 100 inclusive. We also use `//` here, to add comments to the program:

```
[  sumcubes:=0:   //  set sumcubes to zero initially
⎡   for i from 50 to 100 step 2 do  // now press SHIFT AND ENTER
⎢       sumcubes := sumcubes + i^ 3
⎣   end_for;
```

Note the instruction to press SHIFT and ENTER above; if you fail to do this then you will get an error message as the `for` loop has not been closed. Note also that the command line within the loop has been slightly indented; this makes the whole structure easier to read[1]. To check your understanding of loops, try the following exercises.

Exercise 4.1
Use the `for...do...end_for` structure to output $\cos(n\pi/2)$ for $n = 0, 1, 2, \ldots 10$. □

Exercise 4.2
Find $\displaystyle\sum_{n=1}^{N}(2n-1)^2$ for $N = 5, 6, \ldots, 30$. □

Exercise 4.3
Find in decimal form

$$\sum_{n=1}^{100N}\frac{1}{n} \quad \text{and} \quad \sum_{n=1}^{100N}\frac{1}{n^2}$$

for $N = 1, 2, \ldots, 10$. What happens in each case as $N \to \infty$? □

An alternative to the `from <expr> to <expr> step <expr>` structure involves the use of `while`. As a simple example, the sum of the cubes of the even integers between 50 and 100 inclusive could have been found with the commands

```
[  sumcubes:=0:  j:=50:
⎡   while j <=100 do
⎢      sumcubes := sumcubes + j^ 3 :   j := j + 2 :
⎣   end_while :
[  sumcubes;
```

(here the expression `j<=100` means '`j` is less than or equal to 100'; see the box on p. 33 for more details). You should spend some time checking that you are happy with the use of `while`

[1] Recall from Section 1.3.3 that MuPAD ignores any gaps such as this.

above, and with the `j:=j+2:` command. In a `while` loop, is the condition (in this case `j<=100`) checked *before* or *after* each iteration?

4.1.1 Approximate solutions to equations

In this section the `for..do..end_for` structure is illustrated by looking at Newton's method for finding approximate solutions to equations of the form $f(x) = 0$.

The process depends on the fact that 'nice' functions are 'approximately linear'(the First Year course will give some real meaning to this assertion). Starting with a guess at the solution, a, the equation of the tangent through $(a, f(a))$ is

$$\frac{y - f(a)}{x - a} = f'(a),$$

and this cuts the x-axis at $\hat{a} = a - \dfrac{f(a)}{f'(a)}$. This is (we hope) a better approximation to the root than a.

Say we want to find an approximate solution to

$$f(x) = x^5 - x^{1.33} - 1 = 0$$

and we know that an initial approximation to the solution we are interested in is $a = 1.25$. MuPAD can be used to find a better approximation with the following commands:

```
[  f:=x^5-x^(1.33)-1:
[  a:=1.25:
[  a:=a-(f|x=a)/(diff(f,x)|x=a);
```

(recall the use of `|` in Section 2.1.1). If after typing in these commands you return the cursor to the last line and press enter again you ought to get an even better approximation to the root; indeed, repeating this a few times will give better and better approximations.

However, this is a little clumsy and MuPAD can do all the iterations in one go, so that a solution is found that is accurate to a required number of significant figures:

```
[  f:=x^5-x^(1.33)-1:
[  a:=1.25:  b:=0:
   while abs(b - a) > 10^ (-7) do
      b := a :
      a := a - (f|x = a)/(diff(f, x)|x = a) :
      print(a);
   end_while :
```

Can you see what is happening here? If not, ask a demonstrator to explain the commands to you.

Exercise 4.4
Adapt the example immediately above this exercise to find successive roots to

$$f(x) = 5 - \frac{1}{4}\cos(3x) = x,$$

with a starting value of $a = 5$. Show that the root is 5.24979 to six significant figures. (*Hint*: you may need to use `float` to obtain the numerical approximations.) □

4.2 Conditionals

The `if...end_if;` construction allows us to choose alternative courses of action during the execution of a sequence of commands. The general structure is:

```
if <conditional expression> then
  <statement sequence>
elif <conditional expression> then
  <statement sequence>
else
  <statement sequence>
end_if;
```

where there can be as many `elif..then`s as needed (including none at all) and `else` is also optional.

Here is a simple example:

```
[  x:=-3:
⎡   if x > 2 then
⎢     y := x + 1
⎢   elif x > 0 and x <= 2 then
⎢      y := x
⎢   else
⎢     y := x - 1
⎣   end_if :
[  y;
```

The <*conditional expression*> mentioned above is a MuPAD statement which is either true or false. This will usually be a comparison using one of the items in the following table

`=`	equality
`<`	less than
`>`	greater than
`<>`	not equal to
`<=`	less than or equal to
`>=`	greater than or equal to

or using the logical operators `and`, `or` and `not` (or a combination of both).

You will have the opportunity to practise the `if..end_if` structure in the next section after you have learned about procedures.

4.3 Procedures

MuPAD procedures are sets of commands which are needed to be used repeatedly, often with different parameter values each time. For example, a procedure could be used to define a function, to return a matrix or a graph, or to perform a particular calculation. The following is an example of a simple MuPAD procedure.

```
f2c := proc(x)
 begin
 float(5/9 * (x - 32));
end_proc;
```

Note that, in order to prevent an error message, you will need to press SHIFT and ENTER at the end of each line except for the final one (where you just press ENTER). Also, normally the final output would be suppressed, by replacing `end_proc;` by `end_proc:`.)

This procedure converts a temperature from degrees Fahrenheit to degrees centigrade; if x is the degrees in Fahrenheit and y the degrees in centigrade then the two are related by

$$y = \frac{5}{9}(x - 32).$$

To find what 100°F is in centigrade just type

```
[  f2c(100);
```

to see that the answer is approximately 37.8°C. The beauty of the procedure is that other centigrade values can be found from their Fahrenheit equivalents very easily:

```
[  f2c(0);
[  f2c(32);
[  f2c(80);
```

Looking back to the command lines making up the procedure, the main points to notice are:

- The procedure starts with *`procedure name`*`:=proc()`, followed by `begin`, and ends with `end_proc:` (or `end_proc;`), where in the above example the *`procedure name`* is `f2c`.
- The argument of this procedure is `x`. Procedures may have one or many arguments, or none at all.
- The value returned by the procedure is the value of the last statement, that is the statement immediately preceding `end_proc:` (or `end_proc;`). This 'value' need not be a number; for example it could be a graph, a matrix or any MuPAD expression. In the above example the value returned is $5/9(x - 32)$ and this is what is obtained for `f2c(x)` for the particular `x` that is input.

Exercise 4.5
Write a MuPAD procedure called `c2f` which converts temperatures in degrees centigrade to degrees Fahrenheit. Check your procedure by evaluating `c2f(0)` and `c2f(37.8)`. □

As another example, say we wanted to use a procedure to define in MuPAD the function

$$f(x) = y^3 + \frac{2}{1+y^2} + \cos y, \quad \text{where} \quad y = 5 + 2\sin x.$$

This can be done with the following commands:

```
f := proc(x) local y;
     begin
      y := 5 + 2 * sin(x);
      y^ 3 + 1/(1 + y^ 2) + cos(y);
 end_proc;
```

The one extra feature here is the `local y;` declaration (where the semi-colon at the end is essential). What this means is that the value of `y` outside the procedure is unchanged; you should check this by assigning `y` a value before using the procedure and then checking that its value is unchanged afterwards. It is recommended that you declare any 'internal' identifiers to be *local*, with the `local` declaration.

Having typed in the procedure `f` we can now find f at any x value we choose:

```
[  f(0);
[  float(f(0));
[  float(f(2));
```

Exercise 4.6
Write a procedure which defines the function f given by

$$f(x) = \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{otherwise.} \end{cases}$$

Check your procedure by finding $f(2)$ and $f(-3)$.

Hint: Use the `if..end_if` structure of Section 4.2 within your procedure. □

4.4 Further examples of procedures

In this section you will see some more examples of procedures, illustrating some of the wide variety of problems to which they may be applied. You should make sure that you type out each procedure and think carefully about how it works; the procedures are quite lengthy and complicated and it will take time to understand them fully. Each of the examples revises material from earlier on in the manual, and you are also introduced to a few new commands.

4.4.1 Finding the arithmetic mean of a set of numbers

This section revises the material on lists covered in Section 3.6.2. Say we wanted to find the arithmetic mean of a dataset given as a list of elements. This could be achieved using the following procedure:

```
mean := proc(data) local n, s, i;
  begin
  n := nops(data) :   //checking size of dataset
  s := 0 :
  for i from 1 to n do
     s := s + data[i] :
  end_for :
  s/n;
 end_proc;
```

Hence `mean(L)` is a function that returns the mean of the list `L`:

```
[  mean([1,2,3,4,5,6]);
```

4.4.2 Taylor's theorem

Taylor's theorem tells us how well a function which is sufficiently differentiable can be approximated by polynomials in the neighbourhood of a point. The `taylor` command in MuPAD calculates the Taylor series of a function together with an 'order' of the size of the remainder after the maximum degree polynomial terms which you specify. Thus

```
[  taylor(cos(x),x=0,6);
[  expr(%);
```

gives the Taylor expansion of $\cos(x)$ about zero with the remainder at degree six. The `expr` command converts the series to an expression, ie gives the Taylor polynomial without its remainder.

Note that MuPAD doesn't guarantee that the approximating polynomial is very effective; that's the job of the mathematician. However, we are often fortunate, and here's a case where the Taylor polynomials form better and better approximations locally about the expansion point as their degree increases. The following procedure plots the function `f` and its Taylor polynomials about x=`a` up to and including degree `maxdegree` for `left` $\leq x \leq$ `right`. It is fairly self-explanatory, but the use of `$`, to create a sequence of values, is new to you (although you were introduced to `$` in Section 2.3). Type `?$` at the prompt for more information.

```
taypol := proc(f, a, maxdegree, left, right, x) local p, deg, i;
  begin
  for deg from 0 to maxdegree do
    p[deg] := expr(taylor(f(x), x = a, deg + 1)) :
  end_for :
  plot(f(x), p[i] $ i = 0..maxdegree, x = left..right) :
end_proc;
```

For example, try

```
[  f:=u->sin(1/u):
[  taypol(f,0.5,3,0.1,0.7,x);
```

Can you tell which curve is which?

You may have wondered why we needed to pass `x` into the procedure (it is the final argument in `taypol`). This is because, in line 4 of the procedure, `x` is used as what is termed a 'symbolic identifier', and such variables either need to be initialized locally (as happens to `deg` on line 3 of the procedure) or have to be passed as additional arguments in the way shown above. You should not worry about this technicality, as long as you are able to see how it works in the above example.

4.4.3 Euler's method

You have seen how `solve(ode...)` can find analytic solutions to differential equations. However, not all differential equations have analytic solutions and in such cases approximate solutions may be computed using numerical methods such as the following.

The Mean Value Theorem, which holds for a wide class of functions, says that $y(x+h)-y(x) = hy'(x+\theta h)$ for some θ depending on x and h. For suitable functions we can therefore hope that $hy'(x)$ is a reasonable approximation to $y(x+h)-y(x)$. This is the basis of 'Euler's method'.

For an equation in the form $y'(x) = f(x, y(x))$ together with an initial condition $y(a) = \alpha$, say, we must first choose a 'step length' h which gives discrete values of $x_i = a+(i-1)h$, $i = 1, 2, \ldots$. Then set $y_1 = \alpha$ and compute approximations y_i to $y(x_i)$ for $i = 2, 3, \ldots$ using

$$y_i = y_{i-1} + hf(x_{i-1}, y_{i-1}).$$

Here is a MuPAD implementation of Euler's method for $y' = f(x, y)$, with $y(a) = \alpha$, for $a \le x \le nh$, using a step length h. The abscissae (x coordinates) are stored in `abscis` and the solution sequence is stored in `soln`. The procedure uses the MuPAD structure `array` as a way of storing the x_i and y_i; for more details type `?array` at the prompt.

```
eulersmethod := proc(f, a, alpha, h, n) local i, data, abscis, soln;
begin
abscis := array(1..n + 1) :   soln := array(1..n + 1) :
abscis[1] := a :   soln[1] := alpha :
data := [abscis[1], soln[1]] :
for i from 2 to n + 1 do
   soln[i] := soln[i − 1] + h ∗ subs(f, x = abscis[i − 1], y = soln[i − 1]) :
   abscis[i] := a + (i − 1) ∗ h :
   data := data, [abscis[i], soln[i]] :
end_for :
plot(plot :: PointList2d([data]));
end_proc;
```

Hence the approximate solution to

$$\frac{dy}{dx} = -xy, \quad y(0) = 1$$

for $0 \le x \le 1$ using a step size $h = 0.1$ can be plotted as follows:

```
[eulersmethod(-x*y,0,1,0.1,10);
```

4.4.4 Euclid's algorithm

Euclid's algorithm for finding the greatest common divisor of two natural numbers is essentially based on the observation that if $a > b$ then $\gcd(a, b) = \gcd(a - b, b)$. Things can be improved by taking off as much b as possible; by using the `trunc` command which truncates any number to the next nearest integer towards zero the following elegant procedure is constructed:

```
euclid := proc(a, b)
begin
if b = 0 then a else euclid(b, a − b ∗ trunc(a/b))
end_if
end_proc;
```

Note that this procedure calls itself – this is termed *recursion* and is available in MuPAD (but not all programming languages).

4.4.5 A simple matrix procedure

You will be looking at a more in-depth linear algebra problem with MuPAD next term, so we only present a simple example procedure here. It is a procedure to create an $n \times n$ matrix with 0's on the diagonal, 1's everywhere below the diagonal and -1's everywhere above the diagonal. This example will make more sense if you have already worked through Appendix A.

```
mat := proc(n)local A, i, j;
begin
A := matrix(n, n) :
for i from 1 to n do
 for j from 1 to n do
    if i > j then A[i, j] := 1
    elif i < j then A[i, j] := −1
    else A[i, j] := 0
    end_if :
 end_for :
end_for :
A :
end_proc;
```

```
[ mat(4);
```

4.5 Debugging

When you have written a program, you should not be too surprised if it does not work the first time you try to run it. The program may either crash completely or may give the wrong answer.

- **If the program crashes**, you should try to locate the line where it dies. If this is not self-evident, you should substitute semi-colons for colons to execute commands. The program will then print out all intermediate calculations which should make it easier to find the offending line. (This is not true, however, if a program contains procedures; only the result of the last evaluation of a procedure will be printed irrespective of the use of colons or semi-colons within the procedure.) You should then try to work out what is wrong. Remember that the error is very often just *above* the line with the apparent error. Three common syntax errors that can be difficult to spot are (a) to forget the colon/semi-colon after the command, (b) to use `=` rather than `:=`, and (c) to type the letter O instead of the number zero. MuPAD's error messages can help, but they often pinpoint symptoms rather than identifying their cause.

- **If the program runs but gives the wrong answer**, it may prove quite difficult to locate and correct the error. The first thing is to convince yourself that your program

is trying to solve the right problem. You should try to run the program for very simple cases, and find the simplest case in which it gives the wrong answer. Remove all code that is not used in that particular calculation; this is relatively easy to do by inserting `//` so that MuPAD ignores everything that follows up to the end of the line. This will usually make it easier to pinpoint the error.

4.6 Further exercises

These are only for if you really have spare time after studying the procedures of this unit.

Exercise 4.7
Write a procedure that gives the rows of Pascal's triangle up to and including the nth row for any given n. □

Exercise 4.8
Write a procedure using `linalg::det` from the `linalg` library to solve any nonsingular 4×4 matrix system using Cramer's rule. (You may need to do a little research to discover what Cramer's rule is.) □

Appendix A

Matrix and Vector Algebra

This appendix may be studied at any time after you have worked through Chapter 1. However, as less explanation is provided here compared with that given for the core material in Chapters 1–4, you might prefer to wait until you feel fairly proficient with MuPAD. We therefore suggest that you work through this appendix any time after you have completed Chapter 3. The material covered here will be useful for the compulsory first project in Hilary Term.

A.1 Vector definition

Vectors in MuPAD are regarded as special matrices of dimension $1 \times n$ or $n \times 1$. The column vector $\mathbf{a} = [2,\ 1,\ 5,\ 9]^T$ can be defined in MuPAD by typing

```
[  a:=matrix([2,1,5,9]);
```

To change the third element to the value 7, type

```
[  a[3]:=7;
```

and then check the result by typing

```
[  a;
```

To input the row vector $\mathbf{b} = [2,\ 1,\ 7,\ 9]$ type

```
[  b:=matrix([[2,1,7,9]]);
```

Vectors can be defined and initialized in various ways; for example a zero vector can be set up with the command

```
[  c:=matrix(4,1);
```

and then the entries may be input one at a time, as shown above. As another example, try the following command, which sets up the column vector $[1,\ 4,\ 9,\ 16]^T$:

```
[  d:=matrix(4,1,i->i^2);
```

A.2 Matrix definition

Matrices can be built up row by row:

```
[  A:=matrix([[1,2,3],[4,5,6]]);
```

with individual entries changed as necessary:

```
[  A[1,2]:=-1:
[  A;
```

More generally, matrices can be defined by something like the following:

```
[  B:=matrix(2,4,(i,j)->abs(i-j));
```

A.3 Rows, columns and submatrices

Suppose that A is a matrix. Then rows i to j and columns k to l can be extracted with the command `A[i..j,k..l];`. This is illustrated by extracting the first row of `B` defined above:

```
[  B[1..1,1..4];
```

or by extracting the submatrix

```
[  B[1..2,1..3];
```

A.4 Vector algebra

Vector algebra follows the usual rules. In the following, it is assumed that **v1** and **v2** are both vectors of (the same) length n, and s is a scalar.

MuPAD command	Function
`v1+v2`	Vector addition
`s*v1`	Scalar multiplication by s
`linalg::scalarProduct(v1,v2)`	Scalar product of **v1** and **v2**
`linalg::crossProduct(v1,v2)`	Cross product of **v1** and **v2** (for $n = 3$ only)

The third and fourth commands above start with `linalg::`, to signify that they come from the `linalg` library of linear algebra functions. Type `?linalg` at the prompt for more details, if interested. The following commands illustrate some of the above:

```
[  v1:=matrix([1,2,3]);
[  v2:=matrix([4,5,6]);
[  v1+v2;
[  2*v2;
[  linalg::scalarProduct(v1,v2);
```

A.5 Matrix algebra

As you will be aware, matrix algebra can only be performed on matrices of the right shapes, and we shall suppose that the operations defined below have a meaning; if the matrices are the wrong shape MuPAD will complain. In the following table, A and B are matrices, I is the identity matrix with the same dimensions as A, $\mathbf{v}$ is a vector and s a scalar.

MuPAD command	Function
`s*A`	Scalar multiplication by s
`A+B`	Matrix addition of A and B
`A*B`	Matrix multiplication of A by B
`A^n`	Matrix powers (integer n)
`1/A` or `A^(-1)`	Matrix inverse (square matrices)
`A-s`	$A - sI$
`A*v`	Multiplication of A by `v`
`linalg::det(A)`	$\det(A)$
`linalg::tr(A)`	$\text{trace}(A)$
`linalg::adjoint(A)`	$\text{adj}(A)$
`linalg::transpose(A)`	A^T

The command `A - s` above is interesting; in this context MuPAD knows that `A` is a matrix and although `s` is a scalar it assumes that we want to subtract `sI` (so that the subtraction is meaningful).

Don't forget that you can obtain help with these commands by typing `?` followed by the command you are interested in; try `?adjoint`, for example.

A.6 Eigenvalues and eigenvectors

Again it is assumed that A is a matrix, and we now assume that it is also square. The commands `linalg::charpoly(A,x)`, `linalg::eigenvalues(A);` and `linalg::eigenvectors(A);` will provide what they promise (although you should note that the command `linalg::eigenvectors(A);` will also produce the eigenvalues of `A` for free). Consult the help pages for full details, and see also Exercise 5.3 below.

A.7 Exercises

Working through the following exercises will give you some practice in the material introduced in this appendix.

Exercise 5.1

Let $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$, $B = \begin{bmatrix} 1 & 3 & 7 \\ 4 & -5 & 0 \end{bmatrix}$ and $C = \begin{bmatrix} 2 & 5 \\ 4 & 6 \\ -1 & 0 \end{bmatrix}$.

Use MuPAD to find (where possible) $3A$, $A - 2B$, $A + 2C$, AC, CA, AB and A^T.

Verify that $(AC)^T = C^T A^T$. □

Exercise 5.2

Consider the set of simultaneous equations

$$3x + y - z = 1,$$

$$5x + y + 2z = 6,$$
$$4x - 2y - 3z = 3.$$

Write this as a system $A\mathbf{x} = \mathbf{b}$, where A is a 3×3 matrix, $\mathbf{x} = [x,\ y,\ z]^T$ and $\mathbf{b}$ is a column vector of length 3. Use MuPAD to invert the matrix A and hence find the solution $\mathbf{x}$ to the simultaneous equations. □

Exercise 5.3
(a) Use MuPAD to find the eigenvalues and corresponding eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix}.$$

(b) Now use MuPAD to find the eigenvalues and eigenvectors of

(i) A^3, (ii) A^{-1}, (iii) $A - 6I$ (where I is the 2×2 identity matrix), (iv) $(A + 3I)^{-1}$.

In each case can you spot the relationship of the eigenvalues and eigenvectors found in (b) to those found in (a)? □

Appendix B

Simplification

This appendix may be studied at any time after Chapter 3.

The main strength of MuPAD is its ability to manipulate mathematical expressions symbolically. For example, in this manual you have seen how to differentiate and integrate expressions, how to evaluate sums, and how to solve equations. This appendix introduces the four commands `expand`, `factor`, `simplify` and `combine` that rearrange mathematical expressions. Some of the examples given here are necessarily rather mundane, so that you gain familiarity with these new commands, but we also revisit some of the earlier material in this manual to see how the MuPAD output may be improved.

The symbolic manipulation of expressions rarely creates results in the form needed by the user. This is because there are infinitely many ways of writing any expression. A user needs to know how to persuade MuPAD to change the form of a particular expression into what is needed. This is more of an art than a science but a minimum requirement is a working knowledge of the commands and techniques introduced here. Usually some ingenuity is also required, and plenty of experience, so the more practice you gain in these important techniques the better.

The fundamental problem is that there are no absolute rules for 'simplifying' expressions, nor, indeed, is there any definition of what it means to 'simplify' an expression. For example, $\cos 2x$ might sometimes be better rewritten as $2\cos^2 x - 1$, or as $1 - 2\sin^2 x$, or as $\cos^2 x - \sin^2 x$; and sometimes $x^3 - y^3$ is better as $(x-y)(x^2+xy+y^2)$ but sometimes it is better left just as it is. Algebraic systems such as MuPAD have to apply fixed rules and these rules may conflict with the requirements of the user.

This brief introduction to simplification should enable you to learn enough to start manipulating the output produced by MuPAD into forms that are suitable for what you require. We start with `expand`.

B.1 expand

The `expand` command does exactly what its name implies. In this section we will look at how it works when applied to polynomials and to trigonometric functions.

The effect of `expand` on polynomial expressions is fairly obvious; for example the expansion of $(x+1)(x+z)^2$ is found by typing

```
[  expand((x+1)*(x+z)^2);
```

When applied to trigonometric functions `expand` uses the sum rules[1] to remove multiple angles, replacing them with powers:

```
[  expand(cos(2*x));
```

(remember that in MuPAD `cos(x)^2` means $\cos^2 x$, etc.) Other examples of the `expand` command are

```
[  expand(sin(x+y));
[  expand(cos(2*x+y));
[  expand(cos(5*x));
```

If in the last example above we wished instead to obtain the partial expansion $\cos 5x = \cos 4x \cos x - \sin 4x \sin x$ then we would need a little trickery:

```
[  c:=expand(cos(x+y));
[  c|y=4*x;
```

MuPAD will also expand hyperbolic expressions in much the same manner; for example try

```
[  expand(sinh(3*x));
```

B.2 factor

For polynomials the `factor` command is in some ways the opposite of `expand`, and once more it does as one would expect:

```
[  factor(x^5-x^4-7*x^3+x^2+6*x);
[  f:=x^4+4*x^3*y-7*x^2*y^2-22*x*y^3+24*y^4;
[  factor(f);
```

In the second example above `f` has been defined first; this is a useful trick with long expressions to check that they are typed correctly before proceeding.

The `factor` command can also be used on rational expressions:

```
[  factor((x^3-y^3)/(x^4-y^4));
```

(where the common factor $(x - y)$ in the numerator and denominator has been cancelled).

Exercise 6.1
Expand the function $f = (x + y + 1)^3$ and then factorise $f - 1$. □

Exercise 6.2
Use MuPAD to show that

$$\sum_{r=1}^{n} r = \frac{1}{2}n(n+1),$$

$$\sum_{r=1}^{n} r^2 = \frac{1}{6}n(n+1)(2n+1),$$

[1] $\cos(x \pm y) = \cos x \cos y \mp \sin x \sin y$ is an example of a sum rule.

$$\sum_{r=1}^{n} r^3 = \frac{1}{4}n^2(n+1)^2.$$

□

B.3 simplify and Simplify

The commands `simplify` are `Simplify` are the most general of MuPAD's simplification commands and are usually the ones to be tried first. However, they are not always the most appropriate commands and results can be unpredictable, so you are advised to keep a copy of the original material. Normally, however, they do produce a simpler result, and they do so by applying certain rules. For more information about these rules, and to see the differences in the two commands, type `?simplify` and `?Simplify` at the prompt. Broadly speaking, `Simplify` is often slower than `simplify`, but much more powerful, and allows finer tuning, through the use of various options.

Positive integer powers of trigonometric and hyperbolic functions are simplified by these rules as far as possible, as illustrated by the following attempt to simplify $\sinh^4 x - \cosh^4 x$:

```
[  simplify((sinh(x))^4-(cosh(x))^4);
```

One of the most useful applications of `simplify` or `Simplify` is when trying to verify that `expr1` = `expr2`, where `expr1` and `expr2` are both expressions. This is achieved with the command `simplify(expr1-expr2)` or `Simplify(expr1-expr2)`, as illustrated by the following attempt to verify that

$$\frac{1+\tanh^2 x}{1-\tanh^2 x} = \cosh 2x :$$

```
[  expr1:=(1+tanh(x)^2)/(1-tanh(x)^2);
[  expr2:=cosh(2*x);
[  simplify(expr1-expr2);
[  Simplify(expr1-expr2);
```

(In this example you should see that `simplify` is unable to establish the required result, but `Simplify` manages to do so.)

The following exercises will give you some practice in the `simplify` and `Simplify` commands. You may also need to use some earlier commands such as `solve`, `subs`, `|` and `diff`.

Exercise 6.3
Use `simplify` or `Simplify` to show that

$$\text{if} \quad y = \frac{1+\sin x}{1+\cos x} \qquad \text{then} \qquad \frac{dy}{dx} = \frac{\cos x + \sin x + 1}{1 + 2\cos x + \cos^2 x};$$

$$\text{if} \quad y = \ln\sqrt{\frac{1+x}{1-x}} \qquad \text{then} \qquad \frac{dy}{dx} = -\frac{1}{-1+x^2};$$

$$\text{if} \quad y = \ln\left(\frac{(1+x)^{1/2}}{(1-x)^{1/3}}\right) \qquad \text{then} \qquad \frac{dy}{dx} = \frac{-5+x}{6(-1+x^2)}.$$

□

Exercise 6.4
Consider the general cubic polynomial

$$f(x) = \frac{1}{3}ax^3 + bx^2 + cx + d$$

where a, b, c and d are real constants. If the stationary points of f are at x_1 and x_2, use MuPAD to show that

$$f(x_1) - f(x_2) = -\frac{a}{6}(x_1 - x_2)^3.$$

□

Exercise 6.5
Show that the function $f = 1/r$, where $r^2 = (x-a)^2 + (y-b)^2 + (z-c)^2$ and a, b and c are constants, is a solution of Laplace's equation

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} = 0.$$

□

B.4 combine

Whilst you saw in Section B.2 that the `factor` command is in some ways the opposite of `expand` for polynomials, in other circumstances it is more the case that `combine` performs this opposite function. The important thing to note is that, when used in the form `combine(expr,option)`, it will combine subexpressions in `expr` using mathematical identities between functions indicated by `option`. The following examples show how it can be used for the two `option` cases `exp` and `sincos`; for more details type `?combine` at the prompt.

```
[  combine(exp(x)*exp(y),exp);
```

```
[  combine(sin(x)^2,sincos);
```

Indeed, one of the very useful applications of `combine` is trigonometric simplification because it transforms powers of sine and cosine into their multiple angle expansions. For example,

```
[  combine(cos(x)^6,sincos);
```

Exercise 6.6
Use MuPAD to find multiple angle forms for $\sin^7 x$, $\cos^3 x$ and $\cosh^4 x$. □

Appendix C

Miscellaneous

C.1 University ICTC Regulations

All use of the computing and network facilities in the Oxford University Computing Services, as well as all other computing and network facilities throughout the University of Oxford and associated Colleges, is subject to certain rules. These rules concern what is considered to be unacceptable behaviour and misuse, as well as what may infringe licence terms or may be otherwise illegal. Note that all use is permitted for bona fide purposes only, and is subject to proper authorization (which may be provided either explicitly or implicitly).

The University regards computer misuse as a serious matter which may warrant disciplinary (or even criminal) proceedings.

The rules by which you have agreed to be bound can be found at the following webpages; it is your responsibility to familiarize yourself with them:

`http://www.admin.ox.ac.uk/statutes/regulations/196-052.shtml`
`http://www.maths.ox.ac.uk/notices/it/rules/`

Misuse of computing and network facilities and unacceptable behaviour include (but are not limited to) the following:

- Attempting to gain unauthorized access to a facility;
- Using someone else's username and/or password;
- Giving your password to someone else to use, and/or disclosing your password to someone else, and/or being otherwise careless with your password;
- Disregarding the privacy of other people's files;
- Generating messages which appear to originate with someone else, or otherwise attempting to impersonate someone else;
- Sending messages which are abusive or a nuisance or otherwise distressing;
- Accessing, storing, creating, distributing or displaying offensive or explicit material, especially in a public place;
- Trying to interfere with someone else's use of the facilities;

- Disregard for 'computer etiquette';
- Sending chain email;
- Being wasteful of computer or network resources;
- Software piracy (including infringement of software licences or copyright provisions);
- Using the facilities for commercial gain without explicit authorization;
- Physically damaging or otherwise interfering with facilities.

C.2 Plagiarism

In Michaelmas Term you are encouraged to co-operate with fellow students in learning to use the system and to use MuPAD. The Hilary Term projects, however, must be your own unaided work.

The description of each project gives references to books covering the relevant mathematics; if you cannot understand some of this then you are free to consult your tutors or others about it, but not about the project itself. You may discuss with the demonstrators and others the techniques described in the Michaelmas Term Students' Guide, and the commands listed in the Hilary Term Students' Guide or found in the MuPAD Help pages. You may ask the course director to clarify any obscurity in the projects.

The projects must be your own unaided work. You will be asked to make a declaration to that effect when you submit them.

Appendix D

Glossary of commands

`|`

Evaluates an expression at a point.

Example: `[ x^2+3*x|x=2;`

---

diff

Differentiates an expression with respect to one or more of its variables.

Examples: `[ diff(x^3+sin(x),x);` `[ diff(x^2*y^3,y);` `[ diff(y^4*sec(x),x$2,y$3);`

---

float

Evaluates an expression using floating-point arithmetic.

Examples: `[ float(ln(2));` `[ DIGITS:=15: float(exp(3));`

---

int

Evaluates indefinite or definite integrals symbolically (use with `evalf` for numerical solutions).

Examples: `[ int(sin(x),x);` `[ int(exp(u),u=0..2);` `[ float(int(exp(x),x=0..2));`

---

limit

Evaluates limits.

Examples: `[ limit(sin(x)/x,x=0);` `[ limit((x^2-4)/(x^2-5*x+6),x=3,Right);`

---

numeric::solve

Finds approximate numerical solutions to equations.

Example: `[ numeric::solve(sin(x)=x^3-5*x^2+4,x=4..5);`

---

plot

Creates a two-dimensional plot.

Examples: `[ plot(x,x=0..5);`

`[ plot(sin(x),cos(x),x=0..PI,Colors=[RGB::Red,RGB::Blue],LegendVisible);`

`[ plot(x,x^2,x=0..2); [ plot(plot::Polar([1-cos(t),t],t=0..2*PI));`

---

solve

Rearranges expressions to make one or more designated variables the subject.

Examples: `[ solve(2*x+3,x);  [ solve({x+y=2,-x+3*y=3},{x,y});`

Solves a given differential equation for an unknown variable.

Example: `[ solve(ode(diff(y(x),x)+y(x)/x=x^2,y(x)));`

Solves recurrence relations

Example: `[ solve(rec(x(n+1)=n+1+x(n),x(n),{x(0)=1}));`

(See also `numeric::solve` above.)

---

subs

Substitutes a value (or values) into an expression.

Examples: `[ subs(x^2+3,x=1);  [ subs(x^2+3*y,x=1,y=3);`

---

sum

Evaluates sums.

Examples: `[ sum(r^2,r=1..n);  [ sum(3*r,r=1..10);  [ sum(1/r^2,r=1..infinity);`

`[ float(sum(1/(k+k^(3/2)),k=1..100));`
