MATHEMATICA MANUAL

FUNDAMENTALS OF DIFFERENTIAL EQUATIONS

EIGHTH EDITION

FUNDAMENTALS OF DIFFERENTIAL EQUATIONS AND BOUNDARY VALUE PROBLEMS

SIXTH EDITION

R. Kent Nagle

Thomas Polaski

Winthrop University

Edward B. Saff

Vanderbilt University

Arthur David Snider

University of South Florida

Addison-Wesley
is an imprint of

PEARSON

ISBN-13: 978-0-321-74838-6
ISBN-10: 0-321-74838-7

Addison-Wesley
is an imprint of

PEARSON

www.pearsonhighered.com

Preface

This manual is designed to be used specifically with the book *Fundamentals of Differential Equations*, Eighth Edition, by Nagle, Saff, and Snider and the companion book *Fundamentals of Differential Equations and Boundary Value Problems*, Sixth Edition, by Nagle, Saff, and Snider.

This manual contains valuable information for incorporating computer technology into the teaching and learning of the theory and applications of ordinary differential equations. Specifically it contains information of how to use the computer algebra system *Mathematica.* It follows as closely as possible the fine *Maple* manual produced for the above text.

Included in this manual is all the information you will need to use *Mathematica* in the instruction of differential equations. (When we refer to *Mathematica* in this manual we are referring specifically to *Mathematica* 8.0. Some of the commands may not work on earlier versions of *Mathematica*, although users of version 7.0 ought not to have any problems.) The first part of the text contains a general introduction on the use of *Mathematica.* A chapter giving demonstrations on the use of *Mathematica* in calculus and differential equations follows this introduction. Then follow fourteen laboratories which comprise the heart of the manual. These labs are designed to aid in the understanding of central concepts studied in a standard introductory course in differential equations. Each lab introduces the concepts and demonstrates how *Mathematica* is used in the context, but then requires the students to become involved in exploratory exercises and questions. The objectives are clearly stated so that the instructor can choose which labs are most appropriate for the course he or she is teaching. The text closes with suggestions for additional projects for students to do, if desired. The additional projects were originally compiled by Douglas B. Meade of the University of South Carolina.

I hope that the information and ideas presented in this manual will be useful and interesting, to you and to your students in differential equations courses. Comments, corrections, and suggestions for improvement in the manual are welcome.

Thomas W. Polaski
Winthrop University
April 2011

Table of Contents

1. Introduction 1

2. Introduction to *Mathematica* 3

- **2.1 The *Mathematica* Environment** 3
- **2.2 Essentials for using *Mathematica*** 6
- **2.3 Functions and Operators** 10
- **2.4 Plotting with *Mathematica*** 15
- **2.5 Programming with *Mathematica*** 20

3. Using *Mathematica* in Calculus and Differential Equations 23

- **3.1 Calculus Examples** 23
- **3.2 Differential Equation Demonstrations** 27

4. *Mathematica* Notebooks 41

- **Introduction** 41
- **Laboratory 1: First-Order Differential Equations: Graphical Analysis Using Direction Fields** 45
- **Laboratory 2: Numerical Solutions by Euler's Method** 51
- **Laboratory 3: First-Order Differential Equations - Analyzing Analytic Solutions** 56
- **Laboratory 4: Picard's Method** 63
- **Laboratory 5: Applications of First-Order Differential Equations** 70
- **Laboratory 6: Further Applications of First-Order Differential Equations** 76
- **Laboratory 7: Higher-Order Differential Equations** 84
- **Laboratory 8: An Application of 2nd Order Equations – The Mass-Spring System** 91
- **Laboratory 9: The Pendulum** 95
- **Laboratory 10: Forced Equations and Resonance** 100
- **Laboratory 11: Laplace Transformations and Projectile Motion** 104
- **Laboratory 12: Analytical and Graphical Analysis of Systems** 113

Laboratory 13: Numerical Techniques on Systems 120

Laboratory 14: Series Solutions 129

5. Additional Project Descriptions 134

5.1 Notes to the Instructor: Group Projects 134

5.2 Design Your Own Project 136

5.3 The ODE of World-Class Sprints 138

5.4 Consecutive Reactions for Batch Reactors 140

5.5 Normal T-Cells (Part 1 of 2) 142

5.6 T-Cells in the Presence of HIV (Part 2 of 2) 144

5.7 Design of an Electrical Network 147

5.8 Hypergeometric Functions 149

Appendix: *Mathematica* Quick Reference 151

1. Introduction

The study of mathematics can be greatly enhanced with the use of technology. Differential equations, in particular, is a subject in which concepts can be more thoroughly explored with the use of technology. The reason for the importance of technology is that differential equations in many instances involve parameters that can be varied. In addition, they usually involve symbolic and numerical calculations, many of which may be intractable without the computer. Moreover they have solutions that should be visualized graphically to understand behaviors.

Among the benefits of using technology in the teaching of differential equations are the following. The student

● can see in easy fashion the behavior of solutions by means of graphs in the form of direction fields and phase planes;

● will come to understand the role that parameters play in differential equations;

● will be able to examine “what-if” situations to discover properties of differential equations;

● will be able to handle computationally intractable problems and generate numeric solutions; and

● will be able to solve various applications of differential equations that involve extensive computations.

This manual is written for the use of the computer algebra system *Mathematica* in differential equations. The version of *Mathematica* used is *Mathematica* 8.0. *Mathematica* contains built-in libraries that have many routines and commands useful in studying differential equations. Information on the use and syntax of these commands and routines is available in a rather extensive and very readable help directory.

When working with *Mathematica*, one is presented with a notebook on which there can be combined text, input, output, and graphics. Each section of this manual is in fact a *Mathematica* notebook. With ease, commands and text can be edited and corrected. It is easy to copy and paste just as in a word processor. This makes it easy for students to do the mathematics in *Mathematica* in the notebook, display the results and the graphs, and then to make comments and summaries of what they have learned.

There are two introductory chapters that detail how to use *Mathematica.* One is a general introductory chapter “Introduction to *Mathematica*” and the other is a more specific introduction “Demonstrations on Using *Mathematica* in Calculus and Differential Equations.” Both of these can be used for reference when the labs are used from the manual. However, a preliminary reading of them together with trial executions of the commands given is very beneficial for what follows in the manual.

Following the introductory chapters are the laboratories, which comprise the heart of the manual. These cover most of the essential topics in an introductory course in differential equations. Some labs are longer than others; some require more work than others. Each lab introduces the concepts and demonstrates how *Mathematica* is used in the context, but then requires the students to become involved in exploratory exercises and questions. It may be appropriate to only use part of a given lab rather than to do all the exercises. The instructor should use discretion on how to best use the labs and modify them as needed for a given situation.

The last part of the manual provides additional project ideas. The projects given represent diverse kinds of applications including chemical engineering, mathematical epidemiology,special functions, electrical circuits, and athletics. A brief overview of each project is given at the beginning. Students are also encouraged in this chapter to devise their own projects and a process for doing this is given.

In summary, the objective of this manual is to help students use technology in the form of *Mathematica* to aid them in understanding and enjoying the study of differential equations. Although some effort must be exerted to learn the basics of *Mathematica* and work is needed to accomplish the labs, the rewards are well worth the effort. Technology can greatly aid in the comprehension of mathematics.

2. Introduction to *Mathematica*

What is *Mathematica*? *Mathematica* is a computer algebra system (CAS). What is a CAS? As the name suggests, it is a computer software program that manipulates sophisticated algebraic symbols and expressions -- saving you the time, the tedium, and the frustration of doing these by hand. But as we shall see, it is much more. It can do some very advanced, difficult calculations, generate impressive graphs, and generate numerical and symbolic solutions to problems that may be intractable by hand. If used properly, a CAS allows you the user to concentrate more on the concepts of mathematics rather than spending time doing endless, time-consuming computations.

A CAS is a software package that works directly with an expression as a basic data structure (compare this to most software packages that require numeric data and perform numeric operations on the data). *Mathematica* is one of several CASs; other major examples are Maple, Derive, MATLAB, and MathCad.

In this introductory section, the rudiments of using *Mathematica* are given. Discussed in this section are the following topics.

1. The *Mathematica* Environment
2. Essentials for Using *Mathematica*
3. Functions and operators
4. Plotting with *Mathematica*
5. Using *Mathematica* Packages
6. Programming with *Mathematica*

2.1 The *Mathematica* Environment

When you use *Mathematica* on the computer, you work in a *Mathematica* notebook. This document was written in a *Mathematica* notebook. It is the means by which you input, output, save, and print information. A *Mathematica* notebook consists of a list of cells, each of a particular type: text, *Mathematica* input, *Mathematica* output, graphics or animation. The limits of each cell are denoted by square brackets which appear in the right margin of the notebook. These cell delimiters are not generally printed. Cells may be removed by left-clicking on the bracket and pressing DEL.

Computing in *Mathematica* is done by executing a cell in the notebook containing input. You execute a cell by placing the cursor within the cell, and pressing SHIFT and ENTER simutlaneously on the QWERTY keypad or by pressing ENTER alone on the numeric keypad. Unless told otherwise, executing an input cell generates a corresponding output cell. The following is an example of an input cell containing a *Mathematica* command and the output which is produced. It is an integration command.

```
Integrate[x^2, x]
```

$$\frac{x^3}{3}$$

Note that the arguments of all *Mathematica* functions are enclosed in square brackets, and that the names of built-in *Mathematica* functions begin with capital letters. *Mathematica* is a case-sensitive language; lower-case letters and upper-case letters are distinct.

There are basically three types of cells in a notebook: (1) inert text, (2) *Mathematica* input, and (3) *Mathematica* output.

1. text or inert input: The cell you are reading now is an example of text. *Mathematica* commands which occur in such cells are not executable, even if the font used looks like *Mathematica* input. For example, to find **`2*ArcSin[1]`** you must place it in an input cell. You can do this by the standard copy command CTRL-C, then moving the cursor to a space between cells. The cursor will become horizontal when you are between cells. Left-click now to produce a complete horizontal line, then use the standard paste command CTRL-V. *Mathematica* always assumes that new cells are input cells, so we now have an input cell which may be evaluated:

```
2 * ArcSin[1]
```

π

You can also enter text or input by using *palettes*. Palettes contain many specialized symbols and alternative alphabets which may be useful. For our purposes, you will probably only need the Basic Math Assistant palette. To open this palette, go to the Palettes menu at the top of the *Mathematica* window. Clicking on Basic Math Assistant will open a window at the right of your screen. Left-clicking on an entry in the palette places it in the notebook at the cursor's current position. Some palette entries like $\square^{\square}$call for your input. For example, to type x^2 ,you first click on $\square^{\square}$. The base of this expresssion should be filled. Type x to fill the base, then TAB, then 2. Hitting the right arrow key will return you to standard text entry mode.

2. *Mathematica* input**:** Input is a mathematical statement you want *Mathematica* to evaluate. The input text is in **`bold Courier New`** font. *Mathematica* evaluates the command when you hit SHIFT and ENTER simultaneously on the QWERTY keypad or by pressing ENTER alone on the numeric keypad. You can also enter input commands using palettes, so expressions like

$\boldsymbol{\alpha + \beta}$

$\alpha + \beta$

are possible.

3. *Mathematica* output**:** Output, by default, is in `Courier New` font.

The menus displayed at the top of the *Mathematica* window contain pre-programmed commands that can enhance and expedite the creation of a *Mathematica* notebook. A description of some of the options is provided. Other commands will be addressed when the need arises.

■ The File menu

New:	Selecting "Notebook (.nb)" creates a new, untitled *Mathematica* notebook.
Open...:	Opens an existing *Mathematica* notebook.
Close:	Closes the current *Mathematica* notebook.
Save:	Saves the current *Mathematica* notebook.
Print...:	Prints the current *Mathematica* notebook.
Exit:	Exits *Mathematica*. You will prompted if you wish to save unsaved work.

In addition, a list of recently opened notebooks appears in this menu for quick re-opening.

■ The Edit menu

Undo:	Undoes the last typing sequence.
Cut:	Removes selected text and copies it to the Clipboard.
Copy:	Copies selected text to the Clipboard.
Paste:	Inserts the contents of the Clipboard at the cursor.
Find...:	Searches the current notebook for matches to the text contained in the "Search for:" field of the dialog box. Also allows for replacement of the selected text.

■ The Cell menu

Delete All Output:	Deletes all cells in the current notebook that have been produced as output.

■ The Evaluation menu

Abort Evaluation:	Aborts the current evaluation.

■ The Help menu

If you need **help**, the on-line help for *Mathematica* is very good -- once you learn how to use it. The help pages describe the syntax of each command, a brief description of the algorithm used to implement the command, and a few examples illustrating the use of the command. The examples can be very helpful. One can often evaluate a cell in a help workbook and then cut-and-paste it into the current notebook, confirm that the example works as advertised, then modify the arguments to answer the question at hand.

You can get help while in a notebook without using the Help menu. For example, to obtain help on the command **`Plot,`** type **`? Plot`** in an input cell and evaluate.

```
? Plot
```

Plot[f, {x, x_{min}, x_{max}}] generates a plot of f as a function of x from x_{min} to x_{max}.
Plot[{f_1, f_2, …}, {x, x_{min}, x_{max}}] plots several functions f_i. ≫

Alternatively, you can go the Help menu on top of the screen and select one of these options and select either Documentation Center or "Find Selected Function." To search for a particular *Mathematica* function, type it in the dialog box. An index of functions is also available.

2.2 Essentials for using *Mathematica*

At the beginning of each *Mathematica* session, all variables have no value attached to them. It is a good idea to clear variables as you finish using them, or to restart *Mathematica* after saving important. These actions help you to avoid unintentionally using a variable name that has previously been given a value, and future operations can be performed without risk that previous definitions and assigned constants will interfere with the processing. Clearing variables is done with the **`Clear[]`** command.

Mathematica is a computer language; it cannot read your mind. *Mathematica* does follow your instructions to the letter. This means that you need to learn to communicate your needs to *Mathematica* in a way that it understands. Here are some basic symbols and other fundamentals.

Assignments are made with either the = or : = symbols. Using = gives the value of the right hand side (RHS) to the name on the left hand side (LHS). Here is an example.

```
x = 2^3
```

8

```
x
```

8

Ending a *Mathematica* command with a **semicolon** (;) performs the command and suppresses the output display. Thus:

```
x = 5;
```

produces no output, but the variable x has still been assigned the value 5.

```
x
```

5

The := symbol is used when you want to define a function which has yet to be evaluated. So defining the function f as

```
Clear[x];
f[x] = x^3
```

x^3

will not use the rule for other variables:

```
f[y]
```

f[y]

Note the difference in the output when the := symbol is used in concert with a variable name x_:

```
f[x_] := x^3;
f[y]
```

y^3

In doing calculations, you will often need to use previous results that you have got. In *Mathematica*, a single **percentage sign**% always stands for your last result. The double percentage sign %% refers to the second previously executed command, the triple percentage sign %%% refers to the third, and so on. The following demonstrates the use of percentage signs.

```
11;

8 + %

19

% * %%

209

(%%% - %) / %%

  198
- ———
  19

(% * 10) + 55

  935
- ———
  19
```

If we change the initial value of 11 to say 6 and execute, *Mathematica* does not automatically adjust other terms even if those statements contain a reference to the changed variable. The only means of re-evaluating terms is by re-executing them and the order in which statements are executed determines the final results.

If input/output names are shown, you can also reference any previous output as %n, where n is the number of the output line. This operation must be done with care, since the output number may easily be changed if you repeat a sequence of operations.

Mathematica is **case sensitive**-- that is, the names x and X are different, pi and Pi are not the same.

Set braces { } enclose unordered sets of objects or ordered lists of objects.

Square brackets [ ] enclose the arguments to *Mathematica* functions.

Parentheses ( ) alter the order of operations in *Mathematica* commands.

Parentheses with asterisks (* *) enclose material (such as comments) in input cells which will not be evaluated.

The question mark ? is used to get help on commands.

The **standard mathematical functions** are denoted by their standard names: +(plus) , - (minus) , * (times) , / (divided by) , ^ (raised to the power) , `Sin`, `Cos`, `Tan`, `Abs`, and `Sqrt`.

Common constants are predefined in *Mathematica*, their names are: `Pi`, `E`, `I`, and `Infinity`. (Yes, they have to be capitalized exactly like this.) These symbols may be entered from the Basic Math Assistant palette as π, e, i, and ∞, and are given in output in those forms.

Unless specified otherwise, all names are assumed to be complex-valued. This is not often a problem, but there are times when this can lead to surprising results.

Let's get the idea of how *Mathematica* works by letting it perform some arithmetic.

In the following lines, use paper and pencil to first predict what you think *Mathematica* will do. Then execute the command and see what actually happens!

```
x = 4 * 6 + 12 / 6 - 1
```

25

```
(-3)^3
```

-27

```
Abs[%]
```

27

```
Pi
```

π

```
v = Sin[Pi / 4]
```

$\frac{1}{\sqrt{2}}$

```
w = v^2
```

$\frac{1}{2}$

```
v
```

$\frac{1}{\sqrt{2}}$

```
Tan[-Pi / 2]
```

ComplexInfinity

```
3 / (5 - Sqrt[x])
```

Power::infy: Infinite expression $\frac{1}{0}$ encountered. ≫

ComplexInfinity

Now enter some commands of your own! Go back to the tangent calculation above and change it to compute the tangent of $\pi/3$.

Now for some slightly more impressive computations. Note what the addition of **//N** does to a command.

```
2^100
```

1 267 650 600 228 229 401 496 703 205 376

```
2^100 // N
```

1.26765×10^{30}

```
N[Pi, 250]

3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348 ⋱
 8253421170679821480865132823066470938446095505822317253594081284811174502841027019385211 ⋱
 0555964462294895493038196442881097566593344612847564823378678316527120190 9
```

```
400!

64 034 522 846 623 895 262 347 970 319 503 005 850 702 583 026 002 959 458 684 445 942 802 397 169 186 831 ⋱
 436 278 478 647 463 264 676 294 350 575 035 856 810 848 298 162 883 517 435 228 961 988 646 802 997 937 ⋱
 341 654 150 838 162 426 461 942 352 307 046 244 325 015 114 448 670 890 662 773 914 918 117 331 955 996 ⋱
 440 709 549 671 345 290 477 020 322 434 911 210 797 593 280 795 101 545 372 667 251 627 877 890 009 349 ⋱
 763 765 710 326 350 331 533 965 349 868 386 831 339 352 024 373 788 157 786 791 506 311 858 702 618 270 ⋱
 169 819 740 062 983 025 308 591 298 346 162 272 304 558 339 520 759 611 505 302 236 086 810 433 297 255 ⋱
 194 852 674 432 232 438 669 948 422 404 232 599 805 551 610 635 942 376 961 399 231 917 134 063 858 996 ⋱
 537 970 147 827 206 606 320 217 379 472 010 321 356 624 613 809 077 942 304 597 360 699 567 595 836 096 ⋱
 158 715 129 913 822 286 578 579 549 361 617 654 480 453 222 007 825 818 400 848 436 415 591 229 454 275 ⋱
 384 803 558 374 518 022 675 900 061 399 560 145 595 206 127 211 192 918 105 032 491 008 000 000 000 000 ⋱
 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 000 ⋱
 000 000 000
```

400! // N

$6.403452284662390 \times 10^{868}$

```
Clear[x];

f = Sin[2 * x]

Sin[2 x]
```

g = 4 * x^2

$4 x^2$

```
h = .25 * Cos[8 * x]

0.25 Cos[8 x]
```

p = f * g

$4 x^2$ Sin[2 x]

```
u1 = f * h

0.25 Cos[8 x] Sin[2 x]

u2 = f + h

0.25 Cos[8 x] + Sin[2 x]

Clear[x, f, g, h, p, u1, u2]
```

2.3 Functions and Operators

Built into *Mathematica* are the basic functions, relational operators, and logical operators. They are denoted as follows.

Trigonometric: **`Sin[]`**, **`ArcSin[]`**, **`Cos[x]`**, **`ArcCos[]`**, **`Tan[]`**, **`ArcTan[]`**, **`Csc[]`**, **`ArcCsc[]`**, **`Sec[]`**, **`ArcSec[]`**, **`Cot[]`**, **`ArcCot[]`**

Relational: <, <= (or ≤), >, >= (or ≥), == (equal), != (or ≠) (not equal)

Logical: **`And`** (**`&&`**), **`Or`** (**`| |`**), **`Not`** (**`!`**)

Transcendental: **`Log[]`** (the natural logarithm function), **`Log[b, ]`**(the logarithm base b) , **`Exp[]`**(the natural exponential function)

Hyperbolic: **`Sinh[]`**, **`ArcSinh[]`**, **`Cosh[x]`**, **`ArcCosh[]`**, **`Tanh[]`**, **`ArcTanh[]`**, **`Csch[]`**, **`ArcCsch[]`**, **`Sech[]`**, **`ArcSech[]`**, **`Coth[]`**, **`ArcCoth[]`**

There are many other functions as well.

Functions can be composed using the **`Composition`** command. For example:

```
Composition[Cos, Sin][x]

Cos[Sin[x]]

Composition[Cos, Sin][Pi]

1
```

Repeated compositions can be performed as shown. The **`Nest`** command allows you to compose a function with itself easily.

```
Composition[Cos, Sin, Cos][x]

Cos[Sin[Cos[x]]]

Exp[Composition[Tan, Sin][x]]
```

$e^{\mathrm{Tan}[\mathrm{Sin}[x]]}$

```
Composition[Cos, Cos][x]

Cos[Cos[x]]

Nest[Cos, x, 2]

Cos[Cos[x]]
```

As mentioned in Section 2, assigning names to expressions is done by using the equal (=) or the "colon equal" (: =). Once a name is assigned to an expression, that name can be used anywhere within the notebook when you want to refer to the expression. Consider the following assignment.

```
f = x^2
```

x^2

We can use the **`ReplaceAll`** command (abbreviated **`/.`**) to substitute the value of $x = 4$ into the expression for *f*. The → symbol is input by using the minus sign then the greater than sign.

```
f /. x → 4
```

```
16
```

We can declare *f* to be a function by using the := command in conjunction with the blank (input as an underscore _) after the argument x. The blank tells *Mathematica* to see f as a pattern which would apply to any argument, not just x.

```
Clear[f];
f[x_] := x^2
```

Now we can easily evaluate as we would any function. Consider for example

```
f[3]
f[10]
f[x]
f[y]
f[x + y]
```

```
9
```

```
100
```

x^2

y^2

$(x + y)^2$

We can easily consider functions with more variables as shown here:

```
g[x_, y_] := x * Sqrt[y]
h[x_, y_, z_] := Abs[Log[x] * Cos[y] * Exp[z]]
```

We then invoke a function by name and respective input variables

```
g[5, 17]
```

$5\sqrt{17}$

```
g[5.0, 17]
```

```
20.6155
```

```
g[5, 17] // N
```

```
20.6155
```

```
N[g[5, 17]]
```

```
20.6155
```

where the data type of the input determines the form of the computed result. For example, the function *g(5, 17)* has two integer inputs, 5 and 17, and because of this *Mathematica* performed exact computations. Evaluation of *g(5.0, 17)* demonstrates that floating-point data types 5.0 and 17.0 tell *Mathematica* to perform approximate calculations while using **N** (either as a function or as a postfix **//N**) always results in approximate output. By default, floating-point results are approximated to 6 digits (not decimal places!) but can be adjusted to any positive integer by the **N** command as shown:

```
N[g[5, 17], 4]
```

```
20.62
```

Remember that it is a good idea to clear the contents of variables and functions when you are done:

```
Clear[x, f, g, h]
```

Operations can also be performed by *Mathematica* on **sets** and **lists**. In *Mathematica*, a set is an unordered collection of elements enclosed by {}. Consider the following sets:

```
A = {a, b, b, c}
```

```
{a, b, b, c}
```

```
B = {d, a}
```

```
{d, a}
```

Notice that repeated elements are retained. Duplicates are not automatically removed from sets and the elements of sets are not re-ordered in output since *Mathematica* assumes that there is an order among the elements of a set. To counteract this assumption we can use set operations. The set of elements that are in A or B can be obtained by the **`Union`** operator, the elements in common with two sets can be derived by the **`Intersection`** operator, and set difference can be employed by the **`Complement`** operator. The following demonstrates the use of these three operators.

```
Union[A, A]
```

```
{a, b, c}
```

```
Union[A, B]
```

```
{a, b, c, d}
```

```
Intersection[A, B]
```

```
{a}
```

```
Complement[A, A]
```

```
{}
```

```
Complement[A, B]
```

```
{b, c}
```

Consider the following list of numbers:

```
A = {3, 13, 6, 6, 11}
```

```
{3, 13, 6, 6, 11}
```

Any element can be taken from the list by using the list name and element position as shown:

```
A[[1]]
A[[2]]
A[[3]]
A[[4]]
A[[5]]
```

```
3

13

6

6

11
```

The mathematical functions that are built into *Mathematica* are mostly set up to be “listable” so that they act separately on each element of a list. This is, however, not true of all functions in *Mathematica*. Unless you set it up specially, a new function that you introduce may treat lists just as single objects. You can use the **`Map`** command to apply a function like this separately to each element in a list. The **`Map`** operator can be used to apply a command to every element of a list. The following demonstrates this feature:

```
Sqrt[A]
```

$\left\{\sqrt{3}, \sqrt{13}, \sqrt{6}, \sqrt{6}, \sqrt{11}\right\}$

```
Sqrt[A] // N
```

```
{1.73205, 3.60555, 2.44949, 2.44949, 3.31662}
```

```
g[A]
```

```
g[{3, 13, 6, 6, 11}]
```

```
Map[g, A]
```

```
{g[3], g[13], g[6], g[6], g[11]}
```

It will often be useful to take a *Mathematica* expression apart. Suppose that we wish to solve the equation $x^2 = 2$.

```
soln = Solve[x^2 == 2, x]
```

$\left\{\left\{x \to -\sqrt{2}\right\}, \left\{x \to \sqrt{2}\right\}\right\}$

We might need to use either of these solutions in future calculations. The first solution is

```
soln[[1]]
```

$\left\{x \to -\sqrt{2}\right\}$

so we could plug it into a function using the `/.` command (notice that the form of **`soln[[1]]`** matches what we need for this operation.)

```
f[x_] := x^3 - 7 * x + 3
```

```
f[x] /. soln[[1]]
```

$3 + 5\sqrt{2}$

Notice however that simply plugging **`soln[[1]]`** into the function *f* does not work.

```
f[soln[[1]]]
```

$\left\{3 - 7\left(x \to -\sqrt{2}\right) + \left(x \to -\sqrt{2}\right)^3\right\}$

To extract only the number $-\sqrt{2}$ we take

```
soln[[1, 1, 2]]
```

$-\sqrt{2}$

and compute

```
f[soln[[1, 1, 2]]]
```

$3 + 5\sqrt{2}$

We could also evalaute *f* at the other solution using either method. Note how the commands change.

```
f[x] /. soln[[2]]
```

$3 - 5\sqrt{2}$

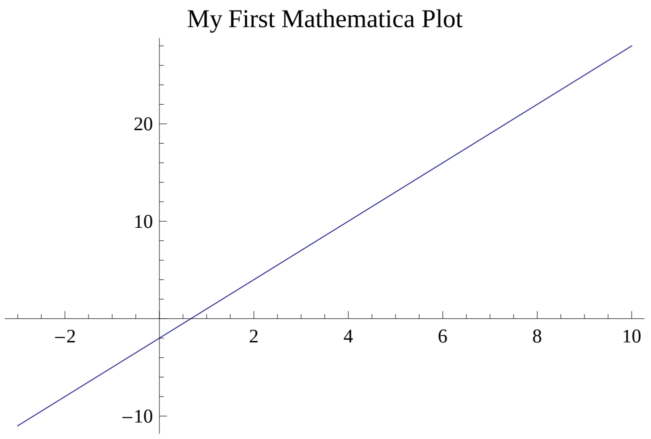
```
f[soln[[2, 1, 2]]]
```

$3 - 5\sqrt{2}$

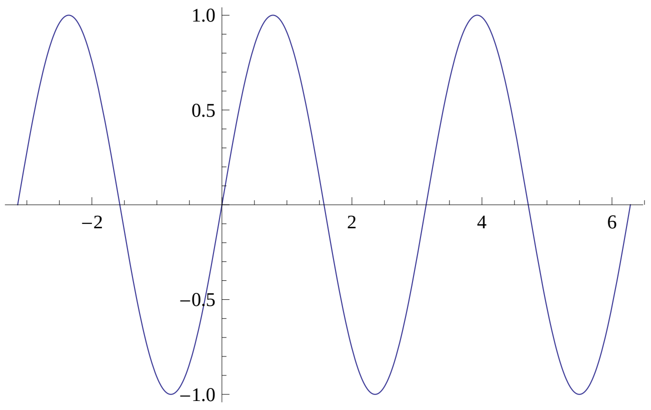
2.4 Plotting with *Mathematica*

Graphs of functions are produced by the **`Plot`** command. In its simplest form, **`Plot`** needs to know the function to be plotted and the range of values for the independent variable. Note that **`{t,a,b}`** is *Mathematica's* way of saying both that the independent variable is *t* and that the plotting interval is [*a*,*b*]. Observe that titles are placed on some of these graphs. Again, try to predict the output before tapping the return key!

```
Plot[3 t – 2, {t, –3, 10}, PlotLabel → "My First Mathematica Plot"]
```



```
f[x_] := Sin[2 x];
(* Here is a sine function! *)
 Plot[f[x], {x, –π, 2 π}]
```



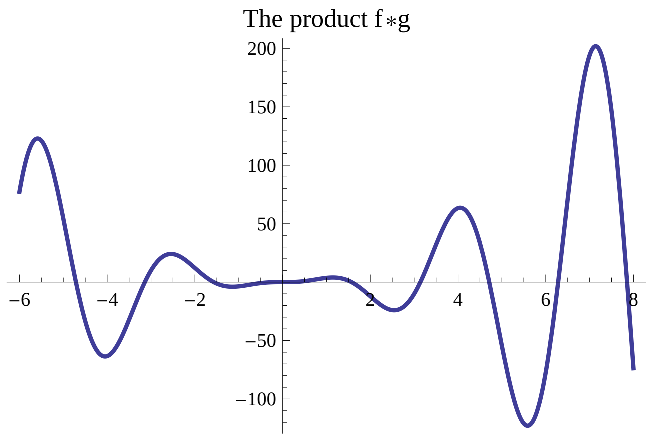
The preceding command has a remark attached. Everything between the (* and *) signs is non-execuable. Notice that ending the **`Plot`** command with a semicolon suppresses output to the screen.

```
 Plot[f[x], {x, –π, 2 π}];
```

```
g[x_] := 4 * x^2;
p[x_] := f[x] * g[x];
```

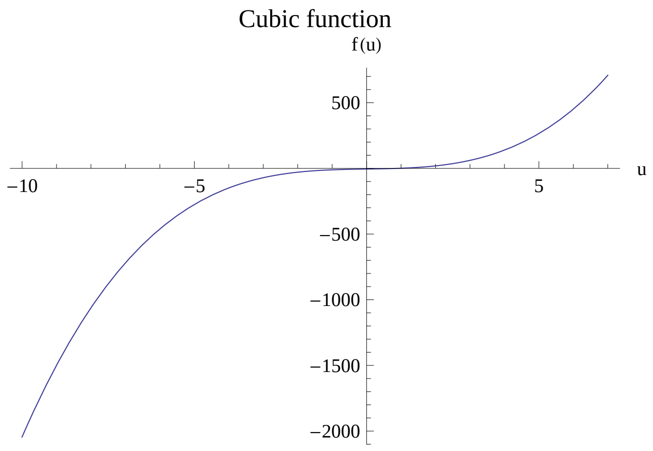
The style of the plot can be changed in various ways, as in the following example.

```
Plot[p[x], {x, -6, 8}, PlotLabel → "The product f*g", PlotStyle → Thick]
```



Consider the following plot.

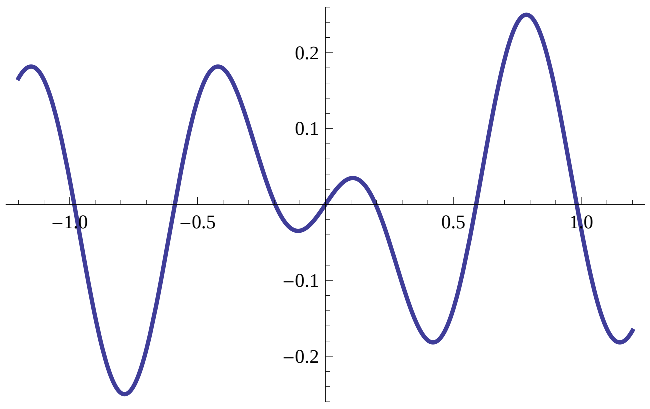
```
Plot[2 u^3 + 4 u - 5, {u, -10, 7}, PlotLabel → "Cubic function", AxesLabel → {"u", "f(u)"}]
```



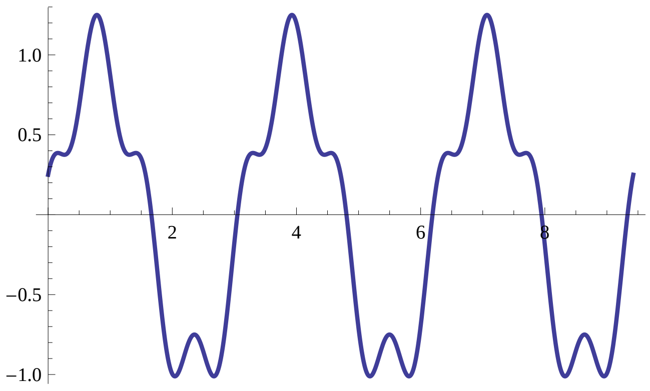
Where does this cubic function cross the *u*-axis? Maybe it would be helpful to cut down the plotting interval from [-10, 7] to [-1, 3], or even narrower, say to [0, 1.5]. Try it. Can you find the *u*-intercept to 2-decimal point accuracy by this process? It is a powerful method, which we call ZOOMING IN.

```
h[x_] := .25 * Cos[8 * x];
u1[x_] := h[x] * f[x];
u2[x_] := f[x] + h[x];
```

```
Plot[u1[x], {x, -1.2, 1.2}, PlotStyle → Thick]
```



```
Plot[u2[x], {x, 0, 3 π}, PlotStyle → Thick]
```

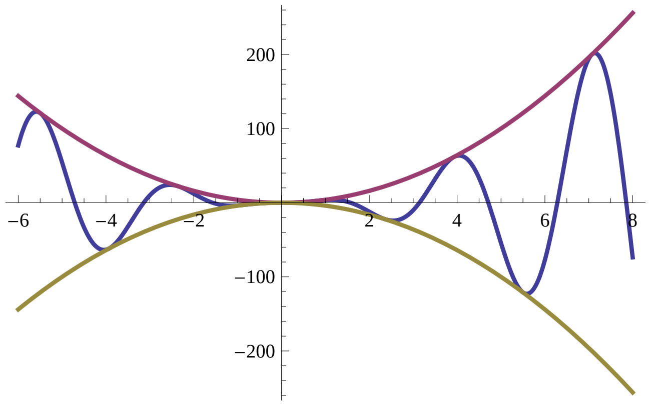


Can you explain why the graphs of *u1* and *u2* look the way they do?

You probably now have a big pile of plots cluttering up your screen. Plots can be reduced in size by selecting the plot, putting the cursor on one of the dots at the corners of the graph, and dragging the cursor on the double arrow that appears. To permanently get rid of plots that you no longer need, click on the cell bracket containing only the plot and depress the delete button. You can delete all of the output in a notebook by using the Delete All Output option under the Cell menu bar.

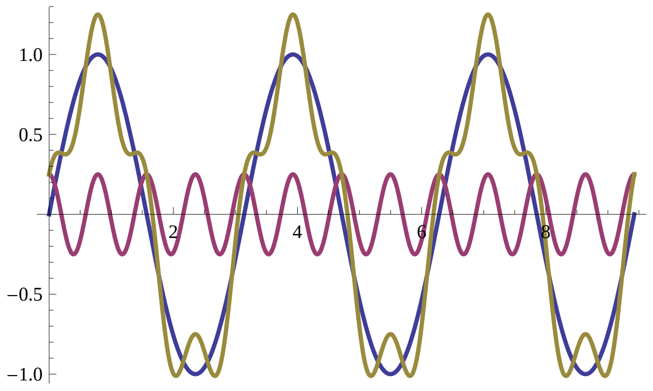
We can also plot many curves simultaneously.

```
Plot[{p[x], g[x], -g[x]}, {x, -6, 8}, PlotStyle → Thick]
```



Can you identify the three different plots? What about the next one? Can you explain the little bumps that appear inside the large dips in the graph of *u2*?

```
Plot[{f[x], h[x], u2[x]}, {x, 0, 3 π}, PlotStyle → Thick]
```

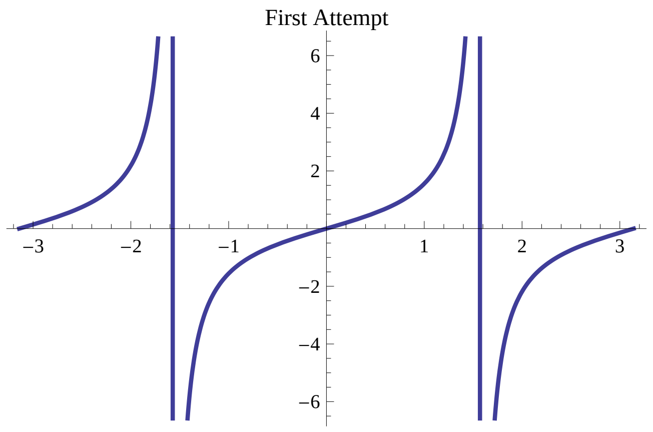


The tangent function is well-known to all of us.

```
f[x_] := Tan[x];
```

Note: This definition completely replaces the previous definition of f . Recall that the graph of *y* = tan(*x*) has vertical asymptotes at all odd multiples of $\pi/2$. Let's see how *Mathematica* handles this.

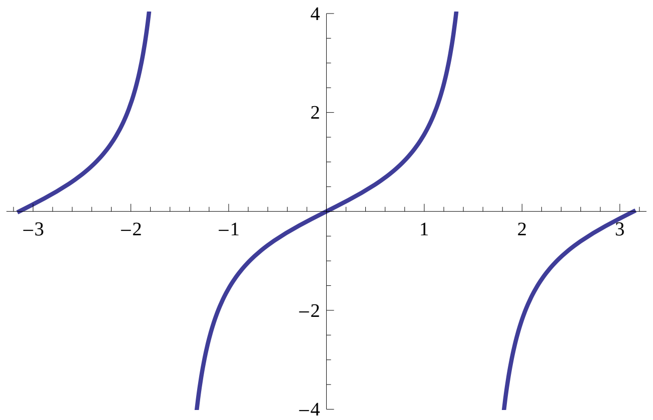
```
Plot[f[x], {x, -π, π}, PlotLabel → "First Attempt", PlotStyle → Thick]
```



Is this what you expected to see? Why does the graph look like this? What can be done to improve the appearance of the plot?

There are two problems. First, we have to tell *Mathematica* that this function is discontinuous. But, there is more. Look at the labels on the vertical axis. The units are larger on the y axis; we want to see the details on a much finer scale. Let's limit the vertical scale to the range from -4 to 4. The optional argument **`PlotRange`** to **`Plot`** explicitly tells *Mathematica* what vertical limits to use on the graph. To handle the discontinuities at $-\pi/2$ and $\pi/2$, we will use another optional argument called **`Exclusions`** to remove those points from the plot.

```
Plot[f[x], {x, -π, π}, Exclusions → {-π / 2, π / 2}, PlotRange → {-4, 4}, PlotStyle → Thick]
```



This technique is very useful for any function that has vertical asymptotes.

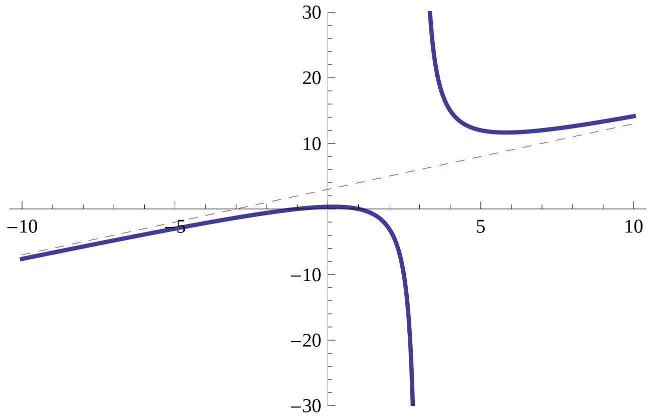
OK. Now it's your turn. Add appropriate optional arguments to the following command to produce a graph from which you can identify the local minimum. (Don't forget the title.)

$$\texttt{Plot}\left[\frac{10 - 4\,t^3}{1 - t^2}, \{t, -3, 5\}, \texttt{PlotStyle} \to \texttt{Thick}\right]$$

Now, one last challenge. Use your graph to estimate the (oblique) asymptote for this function. Check your estimate by graphing the function and the asymptote in the same plot. (Here's an example of what we are seeking, and to show how plots look in a notebook. Note the use of the **`PlotRange`**, **`PlotStyle`** and **`AxisOrigin`** options.)

$$\texttt{g1} = \texttt{Plot}\left[\frac{1 - t^2}{3 - t}, \{t, -10, 10\}, \texttt{Exclusions} \to \{3\}, \texttt{PlotRange} \to \{-30, 30\}, \texttt{PlotStyle} \to \texttt{Thick}\right];$$

```
g2 = Plot[t + 3, {t, -10, 10}, PlotRange → {-30, 30}, PlotStyle → {Thin, Dashed, Red}];
Show[g1, g2, PlotRange → {{-10, 10}, {-30, 30}}, AxesOrigin → {0, 0}]
```



These examples only scratch the surface of *Mathematica's* abilities when it comes to plotting. *Mathematica* can also produce 3-D plots, plots of parametric curves and surfaces, and animated sequences. We will examine some of these in the next section about *Mathematica* packages. The best general source of information is the on-line help for the **`Plot`** command and related packages, which can be accessed through the Help menu or by using the command **`?Plot`**.

2.5 Programming with *Mathematica*

We will create some simple programs in *Mathematica* for use in this manual. The programs will use *repetitive loops* (for loops) and the use of *conditionals* (if - then). There are far more advanced and efficient means to create programs in *Mathematica*, but to introduce them would take us far afield from our task. We call a simple program a *procedure*.

Let's begin with repetitive loops since they will be used often in procedures. The general syntax for a repetitive loop is as follows:

```
For[start, test, incr, body]
```

This command tells *Mathematica* to evaluate the **`start`** command, the repeatedly evaluate the **`body`** and **`incr`** commands until **`test`** fails. Consider as an example of a repetitive loop which outputs 1, 2 and 3 all raised to the 4th power. The repetitive loop would begin as

```
For[n = 1, n ≤ 3
```

```
Syntax::bktmcp : Expression "For[n = 1, n ≤ 3" has no closing "]".

Syntax::sntxi : Incomplete expression; more input is needed.
```

Since the line is not complete, *Mathematica* responds with a warning. The warning statement(s) disappear when the program is valid and complete. Our repetitive loop and output could be in either of the following forms.

```
For[n = 1, n ≤ 3, n++, Print[n^4]];
```

```
1

16

81
```

```
For[n = 1, n ≤ 3, n += 1, Print[n^4]];
```

```
1

16

81
```

The second command allows us control over the amount of the increment. Additional lines can be inserted within the body by separting the commands with a semicolon. For example,

```
For[n = 1, n ≤ 3, n += 1,
  Print[n];
  Print[n^4]];
```

```
1

1

2

16

3

81
```

Notice that inserting returns can make the procedure easier to read.

The general syntax for a conditional statement is as follows:

```
If[condition, t, f]
```

where the command gives **t** if **condition** evaluates to `True` and **f** if it evalautes to `False`. For more options, the **Which** command is used:

Which[$test_1$, $value_1$, $test_2$, $value_2$, …]

where the command evaluates each of the $test_i$ in turn, returning the value of $value_i$ corresponding to the first of the $test_i$ which yields `True`.

As an example of the use of a conditional statement, let's pick 1000 random numbers between 0 and 2. We will say the number is good if it is less than one. We will multiply all those between 0 and 1 together and keep a record of how many between 0 and 1 are selected. We will use the random number generator of *Mathematica* to select the random number. Let's see some of the random numbers generated by this procedure.

```
For[n = 1, n ≤ 5, n++, Print[RandomReal[{0, 2}]]];
```

```
1.97746

0.307963

0.254608

0.332647

1.31522
```

```
good = 0;
bad = 0;
prod = 1;
For[n = 1, n ≤ 100, n++, x = RandomReal[{0, 2}];
  If[x < 1, good = good + 1; prod = prod x, bad = bad + 1];];
Print["Less than one= ", good];
Print["Product= ", prod];
```

```
Less than one= 49
```

Product= 9.43247×10^{-23}

Note that the product is very small as it should be. You can try 1000 numbers and see what the product becomes and see if 1/2 of the numbers chosen are between 0 and 1.

```
Clear[x, good, bad, prod];
```

Finally let's examine more compliacted procedures. We can combine several activities together and call it a procedure. Then whenever we wish to use it we just have to call for it. The general syntax of a *Mathematica* procedure is as follows:

```
Procedure_Name[input_variables] := {
   statement;
     statement;
     ....
    statement};
```

A function is an example of a procedure. We can program the function $f(x) = x^2$ as a procedure as follows:

```
f[x_] := x^2;
```

The procedure can now be invoked by name, in this case *f*, as demonstrated below.

```
f[2]
f[4]
f[a]
f[b]
f[a + b]
```

4

16

a^2

b^2

$(a + b)^2$

In constructing more complicated procedures, it is often useful to declare the scope of the variables used. A local variable is active only within the procedure where it is declared while a variable declared as global retains its value for the entire notebook. The **`Module`** structure is used to declare local variables. Consider this distinction within the following procedure which computes the sum of square integers from 1 to 3.

```
Clear[n];
SumOfSquares[] :=
  Module[{n}, totalsum = 0;
   For[n = 1, n ≤ 3, n++,
    totalsum = totalsum + n^2];
   Return[totalsum]
  ];
```

We invoke this procedure by name,

```
SumOfSquares[]
```

```
14
```

The incremental variable **`n`** was declared local by the programmer. In comparison, the variable **`totalsum`** was by default declared global. Because of this distinction, **`totalsum`** retains its value outside of this procedure, while variable **`n`** does not.

```
totalsum
```

```
14
```

```
n
```

```
n
```