Chapter 1

An Introduction to *Mathematica*

1.1 Introduction

This chapter contains a brief introduction to *Mathematica*. The first few sections will help the reader become familiar with the general syntax of this package. A glossary at the end of the chapter gives a partial listing of the available commands with examples of their usage.

One of the most important aspects of any software package is how readily one is able to access the information from the output of a command and redirect it as input to another. For example, throughout the text we will make extensive use of the differential equation solver of *Mathematica* in various applications, where monitoring the evolution of certain variables is key to our understanding of a physical model. Typically, we would like to graph some or all of the variables we obtain from solving a system of differential equations as time varies, or graph one variable versus another, or integrate and differentiate the output in order to construct physical quantities that have natural interpretations. This chapter is an introduction to how one can accomplish such tasks with simple examples that are similar to the actual circumstances we encounter in the remainder of the text.

The material presented here is intended primarily to be used as reference for the exercises in the upcoming chapters. As a result, during the first reading some of the mathematical language may be unfamiliar. It is hoped that this chapter will become more useful to the reader as one proceeds with the mathematical concepts of the future chapters but returns frequently here to review the relevant syntax.

Using *Mathematica* one is able to

1. Determine roots of polynomials, sum series, and evaluate limits of sequences;
2. Integrate and differentiate symbolically rather complicated expressions;
3. Generate graphics in two and three dimensions;
4. Simplify trigonometric and algebraic expressions;
5. Solve linear and nonlinear differential equations;
6. Determine the Laplace and Fourier transforms of functions;

plus a variety of other mathematical operations. The notation we employ is directed at the usage of *Mathematica* on a SUN workstation. This notation is applicable to other platforms as well, including the Windows version for *Personal Computers* (PCs), after some minor changes that will be clear to the reader.

1.2 A Session in *Mathematica*

There are generally two ways to have *Mathematica* up and running: Either as a stand-alone application (available in DOS and on SUN workstations) or as a Notebook in conjunction with a Windows application such as Windows 3.1 or with xwindows on SUN workstations. The stand-alone version of *Mathematica* is initiated by executing `math.exe` (on PCs) or `math` at the UNIX prompt on a SUN workstation, at which time the package responds with

```
In[1]:=
```

The program is expecting an input from the user. Commands are entered by typing at the terminal and executed by pressing the `Return` (or `Enter`) key.

In the Notebook version of *Mathematica* one either selects the appropriate icon and executes the application (this is the case on PCs and Apple computers) or enters

```
mathematica
```

at the system prompt, as is the case on SUN workstations. After some initialization, a window opens in which one is allowed to enter commands. Commands are entered at the keyboard and executed by pressing the `Shift` and `Return` (or `Enter`) keys **simultaneously**. After the first command is executed `In[1]:=` will appear on the screen.

As mentioned earlier, commands are simply entered at the keyboard. For example, to find the roots of the polynomial

$$f(x) = x^2 - 4x + 3,$$

type

```
Solve[x^2 - 4 x + 3 == 0,x]
```

The program will respond with

```
Out[1]=

     {{x -> 1}, {x -> 3}}
```

that is, 1 and 3 are the roots of the polynomial.

Remark 1.2.1: *Mathematica* distinguishes between $x = y$ and $x == y$. In the first expression, $x = y$, x is assigned the value y, while in the second expression *Mathematica* checks to see if x and y can be equal to each other, that is, it verifies that x and y are compatible objects, and other than that it does not take any action. We will use `==` to define equations.

Remark 1.2.2: Multiplication in *Mathematica* can be entered either by leaving a blank space between the operands (for example, `a x` represents ax) or by putting an asterisk between the terms (that is, `a*x`). If a is a number, however, *Mathematica* interprets the combination of a number adjacent to a variable as multiplication. For instance, `2x` is understood as 2 times x. This convention does not apply to symbols: the expression `ax` is interpreted as the variable whose name is ax and not as the product of the variables a and x.

Remark 1.2.3: We will ignore the prompts `In[]:=` and `Out[]:=` for the rest of this discussion.

The first important feature of *Mathematica* is that this program is **case sensitive**; that is, it distinguishes between `solve` and `Solve` in the first command that we entered. All functions and programs that are **internally** known to *Mathematica* must be capitalized. Thus, `Solve` is a subroutine in *Mathematica* that is capable of finding roots of polynomials, while `solve` does not have a special meaning unless the user has defined it previously. Similarly, *Mathematica* understands that `Sin` is the usual sine function while `sin` is a name of a variable that is unknown in this session.

Functions in *Mathematica* are delimited by [ ] and not by ( ). So `Sin[x]` is the usual sine function while `Sin(x)` is the variable `Sin` multiplied by the variable `x`. The operator `Solve` is an example of a function with two arguments; the first is used to define the equation whose solutions we are interested in, and the second is the variable with respect to which the roots should be computed. All of these arguments are grouped by [ ]. To get a better understanding for the usage of arguments in *Mathematica* let us try the following two examples:

```
b = Solve[x^2 - a x + 3 == 0, x]
```

and

```
c = Solve[x^2 - a x + 3 == 0, a]
```

Do the outputs make sense? In `b` the symbol `x` is the variable with respect to which the roots of the polynomial are determined and `a` is just a parameter, while in `c` the roles of `x` and `a` are reversed.

The above examples show the most important attribute of *Mathematica*: its ability to do symbolic manipulation. In `b` *Mathematica* does not need to know the value for a in order to find the roots of the second-order polynomial. This is just one example among many in which *Mathematica* uses its logical power and is able to find the answer to certain questions. There are limitations to this ability, unfortunately. Try finding the roots of the following polynomial on *Mathematica*:

$$x^5 - ax^4 + 3x^3 + 2x^2 + x - 1 = 0.$$

Mathematica's response is its way of saying that it does not know how to find symbolically the roots of fifth-order polynomials. This answer is in agreement with a celebrated theorem in group theory that confirms our inability to write formulas for roots of general polynomials with degrees larger than or equal to five. On the other hand, by simply drawing the graph of this fifth-order polynomial for a fixed a, we recognize that it has some real roots. *Mathematica* is able to find these roots **numerically**, that is, by approximate methods. To accomplish this one needs to furnish a concrete value of the parameter a in the definition of the polynomial. Try

```
NSolve[x^5 - x^4 + 3 x^3  + 2 x^2 + x -1 == 0 ,x]
```

Mathematica finds all five roots of this polynomial:

```
{{x -> -0.533065 - 0.563943 I},

 {x -> -0.533065 + 0.563943 I}, {x -> 0.425529},

 {x -> 0.820301 - 1.7971 I},

 {x -> 0.820301 + 1.7971 I}}
```

`NSolve` is a program based on a numerical approximation technique that is capable of finding solutions to certain equations; in particular, it is capable of finding roots of polynomials.

In addition to finding roots of polynomials, both `Solve` and `NSolve` are capable of finding zeros of general functions as well as solutions to systems of simultaneous algebraic equations. Consider the function $f(x) = \sin\sqrt{x} - \frac{1}{2}$. Its curve shows that f has a zero in the interval $(0, 2)$. We find its value by using

```
Solve[Sin[Sqrt[x]]-1/2==0,x]
```

Mathematica first gives the warning

```
Solve::ifun:
   Warning: Inverse functions are being used by
     Solve, so some solutions may not be found
```

and then continues with the result

```
         2
        Pi
{{x -> ---}}
        36
```

This function, of course, has infinitely many zeros, as its graph shows. We find the solution to the following system of algebraic equations

$$2x - 3y = a, \quad 3x + 2y = b,$$

by entering

```
Solve[{2 x - 3 y == a, 3 x + 2 y == b}, {x, y}]
```

It results in

```
Out[4]=
       1                      1
{{x -> -- (2 a + 3 b), y -> -- (2 b - 3 a)}}
       13                     13
```

Problems

1. Use `?` (see the next section for more detail) with `Solve` and `NSolve`, and familiarize yourself with the syntax of these commands.
2. Use `Solve` or `NSolve` to determine the roots or zeros of the following expressions.
   (a) $ax^2 + bx + c$
   (b) $ax^3 + bx^2 + cx + d$
   (c) $x^2 + 1$
   (d) $x^3 + 1$
   (e) $x^{1/3} - x + 1$

(f) $\sin x - \frac{1}{3}$

(g) $\sin^2 x - \frac{1}{3}$

(h) $\sin x^2 - \frac{1}{3}$

(i) $\sin x - x$. Familiarize yourself with the syntax of the `FindRoot` command by using `?`. Then use `FindRoot` in this problem.

3. Determine the solution of the following equations using `Solve`, `NSolve` or `FindRoot`.

   (a) $\tan x - 3x + 1$

   (b) The system of algebraic equations

      i. $3x - 2y = 2, \quad x + y = 7$

      ii. $ax - y = 0, \quad x + ay = 1$

      iii. $x^2 - y^2 = 1, \quad x^2 + y^2 = 4$

      iv. $x^3 - y^3 = 1, \quad x^2 - 3xy + y^2 = 8$

      v. $ax + y + z = 1, \quad x - y + 2z = 0, \quad 2x + 3y - z = 2$

      vi. $3x^2 - 4y^2 + 3z = 1, \quad x + y + z = 0, \quad z^3 - x^2 y = -1$

1.3 The Help Command in *Mathematica*

The help command in *Mathematica* is

```
?<command>
```

For example, to gain information about the internal function `Solve` enter

```
?Solve
```

Mathematica responds with (this is the response of version 2.2 of *Mathematica* on a SUN platform)

```
Solve[eqns, vars] attempts to solve an equation or set of equations for the
variables vars. Solve[eqns, vars, elims] attempts to solve the equations for
vars, eliminating the variables elims.
```

In response to

```
?Sol*
```

Mathematica, which treats the asterisk as a wild character, returns

```
SolutionOf   Solve        SolveAlways  SolveDelayed
```

which is a list of all of the commands that have `Sol` as their beginning characters. We can now obtain more information about each command by using the full name with ?. The command

```
??<string>
```

provides even more detailed information about `string` than `?string` does, including the syntax of options that sometimes are not accessed through `?string`. For instance,

```
??Solve
```

returns

```
Solve[eqns, vars] attempts to solve an equation or set of equations for the
variables vars. Solve[eqns, vars, elims] attempts to solve the equations for
vars, eliminating the variables elims

Attributes[Solve] = {Protected}

 Options[Solve] =
   {InverseFunctions -> Automatic, MakeRules -> False,
    Method -> 3, Mode -> Generic, Sort -> True,
     VerifySolutions -> Automatic,  WorkingPrecision -> Infinity}
```

The double query `??` is particularly helpful when the user becomes more familiar with the inner workings of *Mathematica*, since it provides a list of options that one can manipulate in order to obtain a better and more customized output.

1.4 Factoring and Simplification

Some of the elementary operations that *Mathematica* is capable of performing include expanding, factoring, and simplifying expressions. For instance, the command for expanding

$$(a+b+c)^3$$

is

```
Clear[a, b, c];
Expand[(a+b+c)^3]
```

The first command clears any previous values assigned to `a`, `b`, and `c`. *Mathematica* responds with

```
 3      2          2    3      2                   2          2        2    3
a  + 3 a  b + 3 a b  + b  + 3 a  c + 6 a b c + 3 b  c + 3 a c  + 3 b c  + c
```

We can now factor the above expression (assuming that it is stored in Out[4] of the present session) by entering

```
Factor[Out[4]]
```

thereby recovering the original expression. We note that, in place of `Factor[Out[4]]`, we could use

```
Factor[%]
```

to reach the same result if the expression on which `Factor` acts is the latest output of the session.

The operations of `Expand`, `Factor`, and `Simplify` are very useful when we wish to prove the type of identities commonly encountered in elementary algebra. For example, recall the identity

$$(a+b)^2 - (a-b)^2 = 4ab$$

for any two parameters a and b. To verify this identity on *Mathematica* we enter

```
Clear[a,b];
Expand[(a+b)^2 - (a-b)^2]
```

If we repeat the above command but replace `Expand` with `Factor`, we obtain the same result. Now, let us try the previous three commands on

$$a^2 - 2ab + b^2.$$

`Factor` and `Simplify` give us the appropriate alternative expression for $a^2 - 2ab + b^2$, while `Expand` leaves the expression unchanged. Next, we apply these three commands to the expression

$$\frac{1}{x+t} - \frac{1}{x-t}$$

to get a better sense for the range of capabilities of each the above internal functions: `Expand` returns the original expression whereas `Factor` returns

```
      2 t
---------------
(t - x) (t + x)
```

while `Simplify` gives us

```
  2 t
-------
 2    2
t  - x
```

Mathematica is also capable of manipulating trigonometric functions. For example,

```
Simplify[Sin[x+y]-Sin[x-y]]
```

returns the result we expect from elementary trigonometry. To get a sense for the power of *Mathematica*'s `Simplify` command let us try

```
Simplify[((Sin[2 x])^2+(Cos[2 x])^2)^2]
```

Does the answer make sense?

We note that

```
Simplify[Sin[x + y]]
```

returns the same expression as its output, since `Sin[x + y]` is already in its simplest form. However,

```
Simplify[Sin[x +  y] - Sin[x] Cos[y] - Sin[y] Cos[x]]
```

returns the value we expect from this identity.

Problems

1. Verify the following identities in *Mathematica*.
   (a) $\sin 2x = 2 \sin x \cos x$
   (b) $\cos 2x = 2 \cos^2 x - 1$
   (c) $\sin 3x = 4 \sin^3 x - 3 \sin x$
   (d) $\tan(x + y) = \frac{\tan x + \tan y}{1 - \tan x \tan y}$

2. Use *Mathematica* to discover an identity among the given functions.

   (a) $\cot(x+y)$, $\cot x$, and $\cot y$

   (b) $\sin(4x)$, $\sin x$, and $\cos x$

   (c) $\cos x$ and $\cos\frac{x}{2}$

   (d) $\tan 2x$ and $\cos x$

3. Simplify the following expressions.

   (a) $\cos(x+y)+\cos(x-y)$

   (b) $\cos(x+y)-\cos(x-y)$

   (c) $\cos(x+y)+\sin(x+y)$

   (d) $\cos^2 2x-\sin^2 2x$

   (e) $\cos^2 ax-\sin^2 ax$

   (f) $\cos^3 x-\sin^3 x$ (Try `Factor` followed by `Simplify`.)

1.5 Function Definition

We often need to communicate to *Mathematica* that x in $f(x)$ is a variable and should be viewed as such when we wish to determine the function's zeros, plot it, differentiate, or integrate it. The mechanism for defining x as a variable or a placeholder is provided by the underscore character `_`. This character is typed immediately after the letter x the first time x is encountered in the definition of f; the underscore character is *not* used in the subsequent referrals to f. For instance, the function

$$f(x)=x\sin^2 x \tag{1.1}$$

is defined by

```
Clear[x, f];
f[x_] = x (Sin[x])^2
```

Once $f(x)$ is defined in *Mathematica* we may refer to it by name in subsequent operations. For example,

```
Solve[f[x]==0,x]
```

requests that *Mathematica* find the roots of (1.1). Its response shows that f has a triple zero at $x=0$. To see that the argument of f is a dummy variable, let us try

```
Solve[f[t]==0,t]
```

which leads to the same conclusion.

It is also possible to define functions of more than one variable in *Mathematica*. For example, the second-order polynomial $g(x)=x^2-ax+3$ is defined by

```
Clear[a,x,g]
g[x_,a_] = x^2 - a x + 3
```

and the operation of finding its roots is accomplished by entering

```
Solve[f[x,a] == 0, x]
```

Many internal functions in *Mathematica* take as input expressions that involve functions. We have already seen the example of f in (1.1) and the internal function `Solve`. Another example arises with `FindRoot`, a variant of the `Solve` command. This internal function computes roots of functions numerically and is preferable to `Solve` in cases where the function f is rather complicated and one must resort to approximate methods to seek its roots. Its syntax requires specifying an initial guess for a root. For instance,

```
FindRoot[f[x], {x, 3}]
```

determines a root of f by applying a root-finding algorithm to f, based on Newton's method, starting the algorithm at the point $x = 3$. *Mathematica* returns

```
{x -> 3.141069642608009}
```

a good (although not a very good) approximation to the exact value π. There are, however, two options available to `FindRoot` that render the approximate value much closer to the exact value (try `??  FindRoot` for the list of options). They are `MaxIterations` and `WorkingPrecision`. When we try

```
FindRoot[f[x], {x, 3}, MaxIterations->50, WorkingPrecision->25]
```

we end up with

```
{x -> 3.14159263763142393348620340802946915 4‘25}
```

which is accurate to eight digits. Some of the trailing digits in the above answer may differ on your machine. The above calculation was performed on a SUN workstation and PC 486 using version 3.0 of *Mathematica*.

The reader may recall that Newton's method is based on the tangent vector approximation of f, which requires differentiating f. In cases where it is cumbersome to compute the derivative of f, the secant method, which replaces the tangent vector with a chord passing through two points on the graph of f, is preferable. One of the options in `FindRoot` allows for specifying two starting points, at which time a variant of the secant method is called upon. For example,

```
FindRoot[f[x] == 0, {x, 3, 4}]
```

uses function evaluations at $x = 3$ and $x = 4$ to start the approximation algorithm.

Problems

1. Define the following functions in *Mathematica* and evaluate them at the specified points.
   (a) $f(x) = \frac{1-x}{1+x}$: $x = 0, 0.5$, and π
   (b) $f(x) = \log(x + \sqrt{1 - x^2})$: $x = 0, 0.1, 0.2$, and 0.3
2. Consider the function f defined in (1.1). Use `FindRoot` to determine an eight-digit accurate approximation to its root at $x = 0$. Start with an initial guess at $x = 1$.
3. $f(x, t, \omega) = \sin(x - \omega t)$. Use `FindRoot` and find a root of $f(1, t, 4)$ starting with $t = 0$. Repeat with $t = 1$.

4. Use `FindRoot` to find a root of the function f. Experiment with different initial guesses and report on whether the value of the root depends on the initial guess.

   (a) $f(x) = x - \frac{2}{x}$
   (b) $f(x) = x^2 - \frac{2}{x}$
   (c) $f(x) = \frac{x^2-1}{x-2} + x$
   (d) $x \sin x + \cos x$

1.6 Differentiation and Integration

Mathematica is able to differentiate and integrate functions symbolically. The derivative of a function such as $x \sin^2 x$ is found by

```
D[x Sin[x]^2,x]
```

while its integral is determined by entering

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

The `D` and `Integrate` commands of *Mathematica* use the basic properties of the differentiation and integration operators, such as the linearity property, to reduce complicated computations to a series of simpler ones. These properties are then combined with elaborate tables of known derivatives and integrals that allow this software to reach its goal successfully. The power of this software is particularly noticed in the case of integration, where we recall from elementary calculus that we often have to resort to methods such as partial fractions, or special substitutions, or integration by parts to reduce the integrand to a manageable expression. For example, to evaluate the integral

$$\int \frac{1}{1+x^4} dx$$

using standard tables, we must first note that $1 + x^4$ factors into

$$(1 - \sqrt{2}x + x^2)(1 + \sqrt{2}x + x^2)$$

and apply the method of partial fraction before using the table of integration. On the other hand,

```
Simplify[Integrate[1/(1+x^4),x]]
```

yields

```
(-2*ArcTan[1 - Sqrt[2]*x] + 2*ArcTan[1 + Sqrt[2]*x] -
    Log[-1 + Sqrt[2]*x - x^2] + Log[1 + Sqrt[2]*x + x^2])/
  (4*Sqrt[2])
```

The definite integral

$$\int_0^1 \frac{1}{1+x^4} dx \tag{1.2}$$

is determined in a similar fashion:

```
Simplify[Integrate[1/(1+x^4), {x, 0, 1}]]
```

which results in

```
(-2 ArcTan[1 - Sqrt[2]] + 2 ArcTan[1 + Sqrt[2]] -

   Log[2 - Sqrt[2]] + Log[2 + Sqrt[2]]) / (4 Sqrt[2])
```

To get a decimal approximation to the above value, we apply the `N` operation to it:

```
N[%]
```

recalling that `%` stands for the previous output. The new output is

```
0.8669729873399109
```

Mathematica is also capable of performing numerical integration. The internal function `NIntegrate` returns an approximate value of the function on which it operates. For example, to evaluate (1.2) numerically we enter

```
NIntegrate[1/(1 + x^4), {x, 0, 1}]
```

in *Mathematica*, which compares well with the result of the numerical value we obtained after evaluating this integral exactly.

In spite our best effort, it is fair to say the class of functions whose anti-derivative we are able to write down explicitly is rather small. If we just started to list functions at random, we could quickly generate functions whose anti-derivatives are either cumbersome to evaluate or actually impossible to express in terms of elementary functions of calculus. Many functions in mathematical physics fall in the latter category, among them e^{-x^2}, $\sin(x^2)$, and $\frac{1}{\sqrt{1-m\sin^2 x}}$. As a result of efforts of many mathematical analysts in the past few hundred years, properties of such anti-derivatives are tabulated, which are now generally available in most computer algebras, including in *Mathematica*. For example, the function

$$f(x) = \int_0^x e^{-t^2}\, dt$$

can be accessed as

```
f[x_] = Integrate[Exp[-t^2], {t, 0, x}]
```

Mathematica responds with

```
(Sqrt[Pi]*Erf[x])/2
```

The internal function `Erf[x]` is called the **error function**. It appears prominently in probability theory, among other branches of mathematics. We can now manipulate f just like any other function defined in *Mathematica*. Another example is

$$g(x) = \int_0^x \sin t^2\, dt.$$

When we enter

```
g[x_] = Integrate[Sin[t^2], {t, 0, x}]
```

we get

```
Sqrt[Pi/2]*FresnelS[Sqrt[2/Pi]*x]
```

where the function `FresnelS` is called the Fresnel Integral, which has applications in the theory of light diffraction. A third example is

$$h(x) = \int_0^1 \frac{1}{\sqrt{1 - x \sin^2 t}}\, dt. \tag{1.3}$$

This time when we try

```
h[x_] = Integrate[1/Sqrt[1 - x Sin[t]^2], {t, 0, 1}]
```

we get

```
EllipticF[1, x]
```

the answer being given in terms of the **Elliptic Function of the First Kind** . We can obtain numerical values and graphical data from this function as before. For example, let us try

```
h[0.1]
```

Mathematica responds with

```
1.014120373414132
```

where a numerical approximation to the integral in h is obtained.

Remark 1.6.1: The function h defined in (1.3) appears naturally in the context of the period of vibration of a nonlinear pendulum.

Let us proceed a bit further with the last example. Since h defines a function in *Mathematica*, we should be able to differentiate it with respect to x. Let $i(x) = h'(x)$, which we evaluate by

```
i[x_] = D[h[x], x]
```

The output is

```
-EllipticE[1, x]/(2*(-1 + x)*x) - EllipticF[1, x]/(2*x) +
  Sin[2]/(4*(-1 + x)*Sqrt[1 - x*Sin[1]^2])
```

To evaluate i at $x = 0.1$:

```
i[0.1]
```

which results in

```
0.1462958973937115
```

Mathematica is capable of determining partial derivatives as well as multiple integrals. Because we will address these topics in detail later in the text, we postpone their treatment in *Mathematica* at this time.

Problems

1. Differentiate the following functions.
   (a) $f(x) = \log \frac{x}{x+1}$
   (b) $f(x) = \sin^3 4x \cos^5 7(x^2 - 2x + 1)$
   (c) $f(x) = x^{x-1}$

2. Combine the differentiation operation with `FindRoot` and determine the extreme points (the maxima and minima) of $f(x) = x \sin^2 x$. Begin by first defining $g(x)$ to be the derivative of this function and then use `FindRoot` to finds its roots.

3. Integrate the following functions.
   (a) $f(x) = \frac{x}{x-1}$
   (b) $f(x) = x \sin x$
   (c) $f(x) = x^2 \sin x$
   (d) $f(x) = x^{10} \sin x$
   (e) $f(x) = e^x \sin x$ (e^x is `Exp[x]` in *Mathematica*)
   (f) $f(x) = \sin^2 x$
   (g) $f(x) = \sin(x^2)$
   (h) $f(t) = te^{t^2}$
   (i) $f(s) = e^{s^2}$

4. Determine the following indefinite integrals.
   (a) $\int_{-\infty}^{\infty} \frac{1}{1+x^2} dx$ (Ans: `Integrate[1/(1+x^2), {x, -Infinity, Infinity}`)
   (b) $\int_0^{\infty} e^{-t^2} dt$
   (c) $\int_0^{\infty} e^{-at^2} dt$, a is a parameter
   (d) $\int_0^{\infty} e^{-st} \sin t \, dt$ (This is the Laplace transform of $\sin t$.)

5. Define the function f by
$$f(x) = \int_0^x \frac{1}{x + t^5} \, dt.$$
   Use *Mathematica* to determine $f'(\frac{1}{2})$. (Ans: -0.0779959)

6. Define the function f by
$$f(x) = \int_0^1 \sin(xt^3) \, dt.$$
   Determine $f'(\frac{1}{2})$. (Ans: 0.237662)

7. Verify the Fundamental Theorem of Calculus in *Mathematica*:
$$\int_a^b f'(t) \, dt = f(b) - f(a), \qquad \frac{d}{dx} \int_0^x f(t) \, dt = f(x).$$

8. Verify the Leibniz's rule of differentiation in *Mathematica*:
$$\frac{d}{dx} \left(\int_{a(x)}^{b(x)} f(x,t) \, dt \right) = f(x, b(x)) - f(x, a(x)) + \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x} \, dt.$$

1.7 Two-Dimensional Graphics

To plot the graph of a function whose definition is explicitly known, say $f(x) = \sin(x)$ on the interval $[0, 2\pi]$, we give the command

```
Plot[Sin[x], {x, 0, 2Pi}]
```

Mathematica draws a graph of this function and scales it automatically. There are many options associated with `Plot` that would enable one to customize the graph. Among these options are `AxesLabel, PlotLabel`, and `Axes`. For instance,

```
Plot[Sin[x], {x, 0, 2Pi}, AxesOrigin -> {Pi/2,0},
     PlotLabel -> "Graph of Sine", AxesLabel -> {"x","y"}]
```

creates a graph with the proper labels.

In applications we often need to plot several graphs on the same screen. For instance, as we will see in the later chapters, the set of functions $f(x) = \sin nx$ represents a set of basic functions in terms of which we expand Fourier series of a large class of functions. Each $\sin nx$, with n a positive integer, represents a mode of oscillation with period $\frac{2\pi}{n}$. By graphing these functions one sees how the different modes compare to one another. There are several ways to draw the graphs of $\sin nx$, with $n = 1$, 2 and 3 on the same screen. One way is to draw the graphs of each mode separately, and then to combine them by using `Show`:

```
a1 = Plot[Sin[x], {x, 0, 2Pi}];
a2 = Plot[Sin[2 x], {x, 0, 2Pi}];
a3 = Plot[Sin[3 x], {x, 0, 2Pi}];
Show[a1, a2, a3]
```

Each one of the first three lines draws a separate graph. The last line combines the three graphs on a new screen.

A second way of getting the same result is to use the following syntax:

```
Plot[Evaluate[Table[Sin[n x], {n, 1, 3}]], {x, 0, 2Pi}]
```

Figure 1.1 shows the output.

To obtain a hardcopy of any graph we produced in *Mathematica*, we need to take into account the specific features of the software and the platform on which it is installed. For example, to get a hardcopy of the above graph in a Notebook session, we must first highlight the cell containing the graph and then select `Print` from the `File Menu`. On the other hand, when the stand-alone version of *Mathematica* is accessed on a SUN workstation, the command `PSPrint` will do the job. For example, assuming that the above graph is stored in Out[24], try

```
PSPrint[Out[24]]
```

or, simply,

```
PSPrint[%]
```

if the graph is the latest output.

Another way to draw the graphs of $\sin nx$ is by using `GraphicsArray`:

```
Show[GraphicsArray[{{a1}, {a2}, {a3}}]]
```

Now, the above three graphs are plotted separately but on the same screen (cf. Figure 1.2). This form of graphics is useful for displaying wave motion.

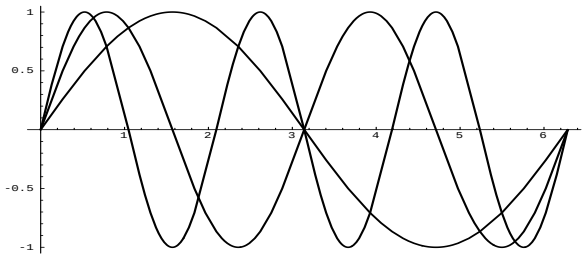

Figure 1.1: The output for `Plot[Evaluate[Table[Sin[n x], {n, 1, 3}]], {x, 0, 2 Pi}]`.

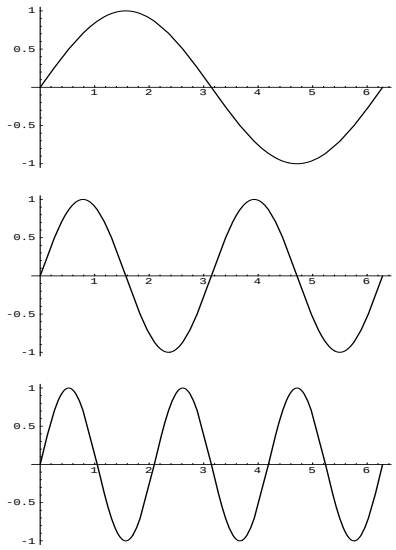

Figure 1.2: The output of the `Show[GraphicsArray[...]]` command.

1.7.1 Curves in the Plane

Curves are the geometric manifestations of particle motions in a domain. Typically in this text curves represent motions of fluid particles in force fields and therefore, are parametrized by time. The position of a particle at time t is represented by a vector $\mathbf{r}(t)$ whose endpoint denotes the location of the particle at t. For example,

$$\mathbf{r}(t) = \langle 2\sin t, 2\cos t \rangle$$

defines a curve in the plane R^2, where the x and y components of each point are $2\sin t$ and $2\cos t$, respectively. Since $x^2 + y^2 = 4$, this curve is a circle of radius 2. To draw its graph, we apply `ParametricPlot` to $\mathbf{r}$:

```
a=ParametricPlot[{2 Sin[t], 2 Cos[t]}, {t, 0, 2Pi}]
```

One of the options of `Show` is `AspectRatio`. Hence,

```
Show[a, AspectRatio -> Automatic]
```

displays a circle with the 1:1 aspect ratio. We have the option of combining the above two commands into one as follows:

```
ParametricPlot[{2 Sin[t], 2 Cos[t]}, {t, 0, 2Pi}, AspectRatio -> Automatic]
```

`ParametricPlot` has several options, among them `PlotPoints` and `PlotStyle`. The `PlotPoints` option is particularly useful when the curve's domain is highly oscillatory because this option designates the number of points at which the parametrization $\mathbf{r}$ is to be evaluated. The default value of `PlotPoints` is 25. Compare the outputs of

```
ParametricPlot[{t, 1/(t+1)+Sin[30 t]}, {t, 0, 2Pi}]
```

and

```
ParametricPlot[{t, 1/(t+1)+Sin[30 t]}, {t, 0, 2Pi}, PlotPoints->50]
```

We will come back to parametrization and `ParametricPlot` in more detail when we review curves, and the concepts of velocity and acceleration, in the context of vector calculus.

Problems

1. Plot the graphs of the following functions.
   (a) $f(x) = \sin(5x)$. What is the period of this function; that is, what is the smallest value of $T > 0$ for which $f(x + T) = f(x)$?
   (b) $g(x) = \sin(2x) + 3\sin(3x)$. What is the period of this function?
   (c) $h(x) = \sin x + \sin\sqrt{2}x$. Draw the graph of this function on the intervals $(0, 5)$, $(0, 10)$, and $(0, 50)$. Do these graphs give any indication as to whether h is periodic or not? Why?
2. Let $f(x) = x^n$. Draw the graph of f for $n = 0, 1, 2, 3, ..., 10$ for $x \in (0, 2)$ on the same screen.
3. Consider the function $f(x) = e^{-x^2}$ with $x \in (-5, 5)$.
   (a) Plot the graph of this function. Use the option `PlotRange -> All` with `Plot` to get the entire range of the function.

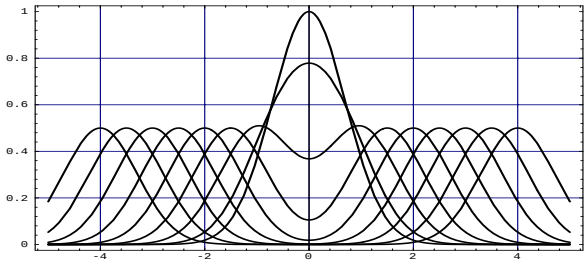

Figure 1.3: A graphics output using the `Frame` and `GridLines` options.

(b) Compare this graph to the graphs of the functions $g(x) = f(x+4)$ and $h(x) = f(\frac{x}{0.5})$ over the same domain. Is there a scale change among these three graphs?

(c) Draw on the same screen the graph of

$$\frac{1}{2}(f(x-2t) + f(x+2t)) \tag{1.4}$$

for t ranging between 0 and 2 at increments of 0.25. Do this part by first defining (1.4) as a new function $g(x,t)$ and then using

```
a=Plot[Evaluate[Table[g[x, t], {t, 0, 2, 0.25}]], {x, -5, 5}]
```

to draw all of the graphs on the same screen. Experiment with the options `GridLines` and `Frame` until you obtain a figure similar to Figure 1.3. Is there a scale change in these graphs?

Expression (1.4) is the formula for the way a wave travels in a string of a guitar when the string is plucked in the shape of $f(\frac{x}{0.5})$ at time zero. Notice that the original "disturbance" (that is, $f(\frac{x}{0.5})$) is broken into two parts, each having amplitude half as much as the original wave and propagating to the left and to the right. Determine the speed of propagation, that is, find how long it takes for the maximum at $x = 0$ at $t = 0$ to reach $x = 1$ and $x = -1$.

4. Plot the graphs of the following curves.

(a) $\mathbf{r}(t) = \langle \sin^2 t, \cos^2 t \rangle$; $t \in (0, 2\pi)$

(b) $\mathbf{r}(t) = \langle \sin^5 t, \cos^5 t \rangle$; $t \in (0, 2\pi)$

(c) $\mathbf{r}(t) = \langle \sin^3 t, \cos^2 t \rangle$; $t \in (0, 2\pi)$

(d) $\mathbf{r}(t) = \langle \sin^3 t, \cos 2t \rangle$; $t \in (0, 2\pi)$

(e) $\mathbf{r}(t) = \langle \sin^5 t, \cos(1+2t) \rangle$; $t \in (0, 2\pi)$

1.8 Three-Dimensional Graphics

The syntax for plotting graphs of surfaces in three dimensions is very similar to two-dimensional graphics. For instance,

```
Plot3D[Sin[x]Cos[y], {x, 0, Pi}, {y, 0, Pi}]
```

produces the surface of the function f defined by $f(x, y) = \sin x \cos y$ in the domain $(0, \pi) \times (0, \pi)$. Similar to `Plot`, `Plot3D` has several options, which we examine in the context of the graph of a "sombrero": The mathematical equation that describes this surface is

$$f(x, y) = \frac{\sin r}{r}, \quad \text{where } r = \sqrt{x^2 + y^2}.$$

First, we define the polar radius r:

```
Clear[r, x, y];
r=Sqrt[x^2+y^2]
```

and then plot the function f by

```
Clear[a];
a=Plot3D[Sin[r]/r, {x, -10, 10}, {y, -10, 10}]
```

We now take two steps to improve the picture on the screen. First, we use `PlotRange` to force *Mathematica* to show us the entire range of the plot:

```
Show[a, PlotRange -> All]
```

The second step is to use `PlotPoints` with `Plot3D` to require *Mathematica* to use more points on the $x-$ and $y-$ axes in its plotting routine:

```
Plot3D[Sin[r]/r, {x, -10, 10}, {y, -10, 10},
     PlotRange -> All, PlotPoints -> 30]
```

The `ParametricPlot3D` is the analogue of `ParametricPlotParametricPlot` for curves whose range is in the three-dimensional space R^3. For example, the graph of a helix whose parametrization is

$$\mathbf{r}(t) = \langle \sin 3t, \cos 3t, t \rangle,$$

with $t \in (0, 2\pi)$, is obtained from

```
ParametricPlot3D[{Sin[3 t], Cos[3 t], t}, {t, 0, 2 Pi}]
```

One can set the aspect ratio of the graph to any desired value using the `AspectRatio` option with either `ParametricPlot3D` or `Show`.

We can also plot the graphs of surfaces with `ParametricPlot3D`. The parametrization of a surface, by definition, requires two independent parameters. For example, the surface whose equation is given by $z = x^2 + y^2$ within the domain $(x, y) \in (-2, 2) \times (-2, 2)$ can also be viewed as the set of points $(x, y, x^2 + y^2)$. Here, the two independent parameters are x and y, each of which takes on values in the interval $(-2, 2)$. `ParametricPlot3D`'s syntax for displaying this surface is

```
ParametricPlot3D[{x, y, x^2 + y^2}, {x, -2, 2}, {y, -2, 2}]
```

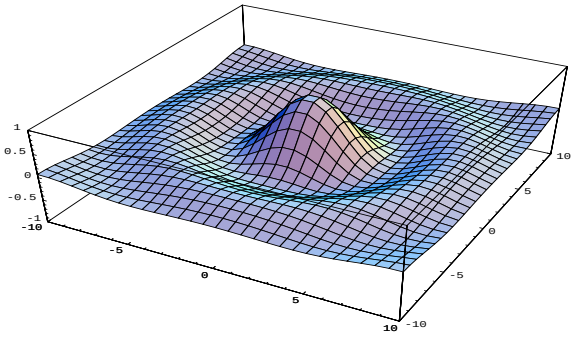

Figure 1.4: The graph of the "sombrero."

The surfaces we have considered thus far have the property that one could write down an explicit formula that relates one of the coordinates of the set of points on the surface to the remaining two. For example, the set of points $(x, y, x^2 + y^2)$, with $(x, y) \in (a, b) \times (c, d)$, can be expressed by the relation $z = x^2 + y^2$. The function $f(x, y) = x^2 + y^2$ is the argument we pass to `Plot3D` in order to draw the graph of this set of points. We now consider examples of surfaces that cannot be readily expressed as $z = f(x, y)$. Many familiar geometric surfaces, among them cylinders, spheres, and tori, are examples of such surfaces. Let us begin with the example of a sphere of radius 1, whose equation is given by

$$x^2 + y^2 + z^2 = 1. \tag{1.5}$$

It is easy to see that (1.5) is equivalent to

$$z = \pm\sqrt{1 - x^2 - y^2}. \tag{1.6}$$

We can now use the two functions in (1.6) with `Plot3D` and combine the resulting surfaces with `Show`. One complication arises because of the domain in (1.6). When we try

```
Plot3D[Sqrt[1-x^2-y^2], {x, -1, 1}, {y, -1, 1}]
```

Mathematica complains about the complex numbers associated with certain values in the domain (such as $x = y = 1$) but still produces a relatively reasonable graph of part of the sphere.

Equations (1.6) describe the surface of the sphere in rectangular coordinates, a coordinate system that is not natural or convenient for plotting the sphere. Instead, we look for the description of this surface in spherical coordinates. In this new coordinate system a point whose rectangular coordinates are x, y, and z is represented by (r, u, v) where

$$x = r \sin u \cos v, \quad y = r \sin u \sin v, \quad z = r \cos u, \tag{1.7}$$

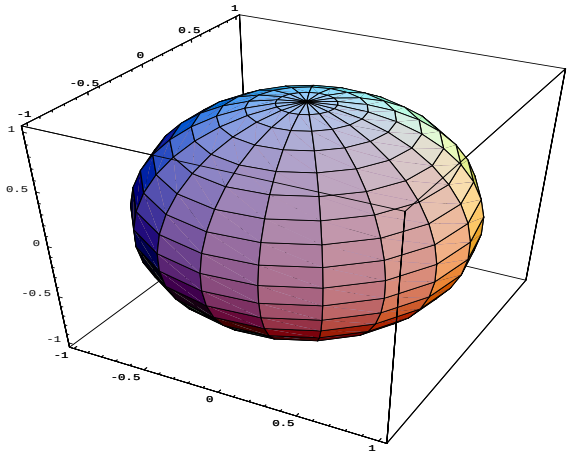

Figure 1.5: The graph of the sphere of radius 1.

where u and v are the standard spherical angles. After substituting (1.7) into (1.6), we note that the equation for the sphere takes the simple form

$$r = 1. \tag{1.8}$$

Equations (1.7) and (1.8), when combined with `ParametricPlot3D`, produce the plot of the unit sphere:

```
ParametricPlot3D[{Sin[u]Cos[v], Sin[u]Sin[v], Cos[u]},
                                 {u, 0, Pi}, {v, 0, 2Pi}]
```

The output of the above line is shown in Figure 1.5.

Many of the familiar shapes and surfaces that we study in mathematics are already programmed in *Mathematica* and are available as built-in functions. To access them we must first enter

```
<< Graphics`Shapes`
```

and read the special library of shapes into our session of *Mathematica*. To get, for example, the graph of a torus, we enter the command

```
Show[Graphics3D[Torus[ ]]]
```

while

```
Show[Graphics3D[MoebiusStrip[2,1,80]]]
```

draws the surface of the Moebius Strip with inner radius 1 and the outer radius 2 using 160 polygons.

Mathematica is capable of rendering animation. In many applications it is possible, and often desirable, to produce a sequence of graphs and put them in motion. For example, consider the

snapshots of a "vibrating string" defined by

$$u(x,t) = \sin x \cos t,$$

with $x \in (0,\pi)$. For each fixed t, the graph of the function $u(\cdot,t)$ is a snapshot of the string. Let us first generate a sequence of these snapshots:

```
Clear[u, x, t];
u[x_, t_] = Sin[x] Cos[t];
Table[Plot[u[x, t], {x, 0, Pi}, PlotRange -> {-1, 1}], {t, 0, 2 Pi, 0.25}]
```

The `PlotRange` option plots all the snapshots over the same vertical range. The period of vibration is 2π (the period of $\cos t$); hence, we let t vary from 0 to 2π, at increments of 0.25. When using *Mathematica*'s Notebook version in Windows 3.1 or on a SUN workstation, we animate the above sequence of graphs by selecting first the cell that contains all of the graphs; next we choose `Cell` from the top menu, from which we select `Animate Selected Graphics`.

The animation of a sequence of surfaces is accomplished in a similar manner. Consider, for example, the function u defined by

$$u(x,y,t) = \sin 3\pi x \sin 2\pi y \cos \pi t, \qquad (x,y) \in (0,1) \times (0,1).$$

Such a function often represents the displacement of an elastic membrane whose boundaries (that is, $x = 0, 1$ and $y = 0, 1$) are constrained not to move. To render an animation of u, first we generate a sequence of its snapshots as follows:

```
Table[Plot3D[Sin[3 Pi x] Sin[2 Pi y] Cos [Pi t], {x, 0, 1}, {y, 0, 1},
            PlotRange -> {-1, 1}], {t, 0, 1, 0.05}]
```

The output is put into animation as before, by first selecting the cell that contains all of the individual snapshots and then selecting `Animate Selected Graphics` from `Cell`.

Problems

1. Draw the graph of the following curves in the specified domain.
   (a) $\mathbf{r}(t) = \langle t, t, t\rangle$; $t \in (0,1)$
   (b) $\mathbf{r}(t) = \langle \frac{t}{12}, \frac{t}{4}, \frac{1}{2+\sin t}\rangle$; $-2\pi < t < 2\pi$
   (c) $\mathbf{r}(t) = \langle e^{-\frac{t}{4}} \sin 3t, e^{-\frac{t}{4}} \cos 3t, \frac{t}{12}\rangle$; $t \in (0, 4\pi)$ (use the `PlotPoints` option of `ParametricPlot3D` to get a graph with better resolution)
   (d) $\mathbf{r}(t) = \langle \sin t, \cos t, \frac{1}{\sqrt{t^2+1}}\rangle$; $t \in (0, 2\pi)$
   (e) $\mathbf{r}(t) = \langle \sinh \frac{t}{6}, \sin(4t), \cosh \frac{t}{6}\rangle$; $0 < t < 4\pi$
   (f) $\mathbf{r}(t) = \langle t + \sin 3t, \frac{1}{t^2+1}\rangle$; $t \in (-2\pi, 2\pi)$
2. Draw the graph of each of the following curves.
   (a) A circle of radius 2 centered at the origin and located in the xy-plane
   (b) A circle of radius 2 centered at the origin and located in the $z = 1$ plane
   (c) The ellipse located in the $xy-$ plane centered at the origin with major and minor axes of 3 and 2, respectively

(d) The curve of intersection of $x^2 + y^2 = 1$ and $z = x$

3. Draw the graph of the following surfaces.

   (a) $z = x^2 + y^2$, with $(x, y) \in (-3, 3) \times (-3, 3)$

   (b) $z = \sqrt{x^2 + y^2}$, with $-3 < x < 3$ and $-3 < y < 3$

   (c) $z = 3x^2 + 4y^2$, in $(-3, 3) \times (-3, 3)$

   (d) $z = \sin(x^2 + y^2)$, with $x \in (-\pi, \pi)$ and $y \in (-\pi, \pi)$

   (e) $z = \sin(\sqrt{x^2 + y^2})$, in $(-\pi, \pi) \times (-\pi, \pi)$

   (f) $z = \sin(x^2 + y^2)\cos(x)$, in $(-\pi, \pi) \times (-\pi, \pi)$

   (g) $z = \sin(x^2 + y^2)\cos(y)$, in $(-\pi, \pi) \times (-\pi, \pi)$

   (h) $z = \frac{\sin\sqrt{x^2+y^2}}{\sqrt{x^2+y^2}}$, in $(-\pi, \pi) \times (-\pi, \pi)$

4. Plot the graph of a unit sphere centered at $(1, -1, 0)$.

5. Plot the graph of a sphere of radius 2 centered at $(-1, 1, 0)$. Combine this graph with the sphere in the previous problem on the same screen.

6. Draw the graph of a cylinder of height 2 and radius 3. (Hint: A cylinder of height h and radius a may be parametrized as $\mathbf{r}(u, v) = \langle a\cos u, a\sin u, v\rangle$, where $0 < u < 2\pi$ and $0 < v < h$.)

7. Draw the graph of a "dumbbell," that is, two spheres connected to a cylinder.

8. Draw the graph of the surface given by

$$z^2 + 9\sqrt{x^2 + y^2} = 9$$

   for $z > 0$.

9. Animate the sequence of curves or surfaces. In each problem t represents time. Find the period of vibration in each case and generate a sequence of snapshots of the curve or surface over an entire period.

   (a) $u(x, t) = \sin 3x \cos t;\ x \in (0, \pi)$

   (b) $u(x, t) = 2\sin 3x \cos t - 3\sin x \cos 2t;\ x \in (0, \pi)$

   (c) $u(x, y, t) = \sin 3\pi x \sin 2\pi y \cos \pi t;\ (x, y) \in (0, 1) \times (0, 1)$

   (d) The graph of a sequence of spheres whose radii at time t are described by $\cos 2\pi t$

1.9 Solving Differential Equations

Mathematica has two internal functions, `DSolve` and `NDSolve`, that are capable of solving special classes of ordinary differential equations. `DSolve` is primarily used to find the exact solution to first-order (nonlinear) equations or linear equations with constant coefficient equations. Here are some examples. Consider the initial value problem

$$v' + v^2 = 0, \quad v(0) = 1.$$

To find a solution to this equation, we enter

```
Clear[a, v, t];
a=DSolve[{v'[t] + v[t]^2 == 0, v[0] == 1}, v[t], t]
```

The reason for the label `a` will become clear shortly. The output is

```
             1
{{v[t] -> -----}}
          1 + t
```

This output is interpreted as a replacement rule, that is, a rule that replaces `v[t]` with

```
  1
-----
1 + t
```

whenever `a` is called upon. For example,

```
v[t] /. a/. t -> 2
```

results in

```
 1
{-}
 3
```

while

```
Plot[v[t] /. a, {t, 0, 3}]
```

plots the graph of `v`. If more than one solution is stored in `a`, `Plot` displays the graph of all solutions.

It is often convenient to define a function `v` using `a`:

```
v[t_] = v[t] /. a[[1]]
```

we can now manipulate the function `v` just like any other function. Assuming v represents the velocity of a particle of unit mass, we determine its kinetic energy during the time interval $t \in (0, 3)$ by

```
1/2*Integrate[v[t]^2, {t, 0, 3}]
```

Similarly, the acceleration of the particle is obtained by differentiating `v[t]` once

```
D[v[t], t]
```

Remark 1.9.1: The inclusion of the independent variable t in `v[t]` is not optional. For example,

```
DSolve[{v' + v^2 == 0, v[0] ==1}, v, t]
```

does not lead to the correct solution of this equation. Also, the operand `==` cannot be replaced by `=`. Here $v' + v^2 = 0$ is an equation and not an assignment; hence, it is necessary to use `==`.

Next, let us consider the differential equation

$$mv' + kv = -mg, \quad v(0) = 0.$$

Here v represents the velocity of an object of mass m falling under the action of gravity and being resisted by a linear frictional force. From

```
Clear[b, v, t];
b=DSolve[{m v'[t] + k v[t] == - m g, v[0] == 0}, v[t], t]
```

we receive

```
                        (k t)/m
             -(g m) + E        g m
{{v[t] -> -(----------------------)}}
                    (k t)/m
                   E        k
```

Assuming k and m are positive, it is clear from the above expression that the terminal (limiting) velocity of the object is $-\frac{mg}{k}$. With $m = 70$, $k = 10$, and $g = 9.8$, we define

```
v[t_] = v[t] /. b[[1]] /. {m -> 70, k -> 10, g -> 9.8}
```

and determine v's terminal velocity by

```
Limit[v[t], t -> Infinity]
```

Mathematica returns

```
-68.59999999999999
```

`DSolve` is an effective tool for solving linear higher order differential equations with constant coefficients. Consider the initial value problem

$$x'' + 3x' + 2x = 0, \quad x(0) = 1, x'(0) = 0.$$

The exact solution of this problem is found by

```
Clear[x, t];
DSolve[{x''[t] + 3 x'[t] + 2 x[t] == 0, x[0] == 1,x'[0] == 0}, x[t], t]
```

whose output is

```
Out[36]=
                     t
           -1 + 2 E
{{x[t] -> ---------}}
              2 t
             E
```

The command `DSolve` works equally well with systems of equations. Consider the initial value problem

$$x' = 2x + 3y, \quad y' = x, \quad x(0) = -2, \quad y(0) = 2.$$

We input

```
b=DSolve[{x'[t] == 2 x[t] + 3 y[t], y'[t] ==  x[t],
          x[0] == -2, y[0] == 2}, {x[t], y[t]}, t]
```

into *Mathematica* and obtain the output

```
Out[38]=
           -2          2
{{x[t] -> --, y[t] -> --}}
           t           t
           E           E
```

In the applications where systems of equations appear, the solution pair $(x(t), y(t))$ is often the parametrization of the path of a fluid particle. To get the graph of this path we enter

```
ParametricPlot[{x[t], y[t]}/. b[[1]], {t, 0, 3}]
```

Methamatica first gives the warning

```
ParametricPlot::ppcom:
   Function {x[t], y[t]} /. b[[1]]
     cannot be compiled; plotting will proceed with
     the uncompiled function.
```

and then proceeds to graph the particle path. To work with compiled functions, first we apply the `Evaluate` command to the solution pair `x[t], y[t]` and then plot the result

```
ParametricPlot[Evaluate[{x[t], y[t]} /. b], {t, 0, 3}]
```

Alternatively, we define a function `f` as the solution pair `x[t], y[t]` and then plot `f`:

```
Clear[f];
f[t_] = {x[t], y[t]} /. b[[1]]
ParametricPlot[Evaluate[f[t]], {t, 0, 3}]
```

The second command in *Mathematica* that is capable of determining solutions of differential equations is `NDSolve`. This program uses a numerical algorithm (based on the standard Runge–Kutta scheme) and solves linear as well as nonlinear systems of differential equations. Consider the forced nonlinear pendulum equation

$$x'' + 0.1x' + \sin x = 0.02 \cos t, \tag{1.9}$$

with initial conditions

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

We have the option of giving (1.9) as a second-order equation to *Mathematica* or as a first-order system. In the first case the syntax is

```
Clear[c, x];
c=NDSolve[{x''[t] + 0.1 x'[t] + Sin[x[t]] == 0.02 Cos[t],
      x[0] == 0, x'[0] == 1}, x, {t, 0, 5}]
```

Mathematica responds with

```
{{x -> InterpolatingFunction[{0., 5.}, <>]}}
```

which states that it has successfully obtained an approximate solution to the above differential equation and has interpolated a curve through the data points (t_i, x_i), where $t_i \in (0, 5)$ are the discretized values chosen by the Runge–Kutta algorithm. We now define a function `x` by

```
x[t_] = x[t] /. c[[1]]
```

To get the graph of the solution x of (1.9), we enter

```
Plot[x[t], {t, 0, 5}]
```

Its value at a point such as $t = 2.15$ is evaluated by

```
x[2.15]
```

which results in 0.8674379209871004. The pair $(x(t), x'(t))$ is plotted by

```
ParametricPlot[Evaluate[{x[t], x'[t]}], {t, 0, 5}]
```

The last statement brings up the point of solving the second-order differential equation (1.9) as a first-order system so that information about x and x' is available simultaneously. To reduce this equation to a first-order system, let $x' = y$, form which we deduce that $y' = -0.1y - \sin x + 0.02\cos t$. Thus, (1.9) is equivalent to

$$x' = y, \quad y' = -0.1y - \sin x + 0.02\cos t, \quad x(0) = 0, \quad y(0) = 1. \tag{1.10}$$

We are now able to determine the approximate solution of (1.9) and plot its particle path in the following manner:

```
sol = NDSolve[{x'[t] == y[t],
               y'[t] == - 0.1 y[t] - Sin[x[t]] + 0.02 Cos[t],
               x[0] == 0, y[0] == 1},
               {x, y}, {t, 0, 10}];
ParametricPlot[Evaluate[{x[t], y[t]} /. sol], {t, 0, 10}]
```

We first receive the output

```
{{x -> InterpolatingFunction[{0., 10.}, <>],

  y -> InterpolatingFunction[{0., 10.}, <>]}}
```

The particle path is then graphed, as shown in Figure 1.6.

`NDSolve` has several important options that help considerably with solving nonlinear differential equations accurately. One of these options is `MaxSteps`. This parameter sets the number of steps that the Runge–Kutta algorithm is allowed to take in discretizing the domain $t \in (0, T)$. If T is too large, then the value of `MaxSteps` (whose default value is 500) must be increased.

Problems

1. Use `DSolve` to solve the following differential equations.
   (a) $x' + 3x = 0$
   (b) $x' + tx = 3e^{-t^2}, \quad x(1) = 2$
   (c) $x' + x^3 = 0$
   (d) $y'' + y = 0, \quad y(0) = 0, y'(0) = 1$
   (e) $x''' + x' + x = 0$

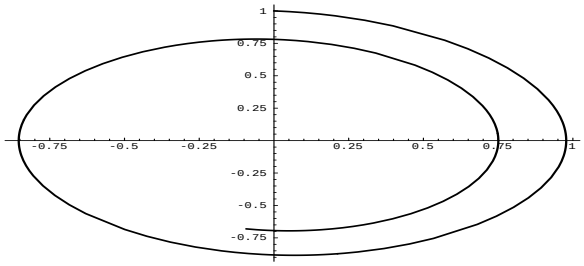

Figure 1.6: The phase plane diagram of the solution to $x'' + 0.1x + \sin x = 0.02\cos t$.

(f) $y'' + y = 0, \quad y(0) = 1, y(1) = -1$

(g) $x'' + x = \sin 2t$

(h) $y'' + y = \sin t$

2. Use NDSolve and draw the trajectories of the following differential systems. Unless otherwise specified, in each case choose a suitable final value for t.

(a) $x'' + x = 0; \quad x(0) = 0, x'(0) = 1$

(b) $y'' + 0.1y' + \sin y = 0; \quad y(0) = 0, y'(0) = 1$

(c) $y'' + 0.1y' + \sin y = 0; \quad y(0) = 0, y'(0) = 3$

(d) $x'' + 0.1x + \sin x = 0.02\cos t; \quad x(0) = 0, x'(0) = 1,\ t \in (0, 100)$

3. Solve each system of differential equations for two sets of initial data $(x(0), y(0)) = (1, 1)$; $(x(0), y(0)) = (2, 2)$. Plot the solutions on the same screen.

(a) $x' = y, \quad y' = -x$

(b) $x' = y, \quad y' = -x - 0.1y$

(c) $x' = y, \quad y' = -x - y$

(d) $x' = y, \quad y' = -x - y^2$.

4. Consider the system

$$x' = y, \quad y' = -\frac{1}{(1 + \epsilon x)^2},$$

with initial data $x(0) = 0$, $y(0) = 1$, and $t \in (0, 3)$. Solve this system with $\epsilon = 0, 0.1, 0.2$, and 1. Plot the graphs of the solutions $(x_\epsilon(t), y_\epsilon(t))$ on the same screen. What seems to be the effect of ϵ on the solutions?

1.10 Vectors, Matrices, and Lists

In *Mathematica* vectors and matrices are entered as lists. For example, the vector $a = \langle -2, 1, 3 \rangle$ is entered as

```
a = {-2, 2, 3}
```

Similarly, the matrix

$$B = \begin{bmatrix} 1 & -1 & 2 \\ 0 & 1 & 0 \\ -1 & 5 & 1 \end{bmatrix}$$

is entered as

```
B = {{1, -1, 2}, {0, 1, 0}, {-1, 5, 1}}
```

We can write B in matrix form by invoking the `MatrixForm` command. Thus,

```
MatrixForm[B]
```

gives

```
1    -1   2

0    1    0

-1   5    1
```

Elements of a list are accessed by putting the subscript of the element between the delimiters `[[...]]`. For example, the first entry of a is `a[[1]]` while the (1,2)-entry of B is `B[[1,2]]`. Also, `B[[1]]` returns

```
{1, -1, 2}
```

which is the first row of `B`.

The length of a list is the number of elements in the list. For example, the vector `a` and the matrix `B` both have lengths equal to 3 as can be checked from `Length[a]` and `Length[B]`.

The standard matrix multiplication is carried out in *Mathematica* by placing a period (.) between the matrices. Thus, to compute the product $A_1 A_2$ of the matrices

$$A_1 = \begin{bmatrix} 2 & -1 \\ -3 & 4 \\ 1 & -1 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

we enter

```
A1 = {{2, -1}, {-3, 4}, {1, -1}};
A2 = {{3, 0, 2}, {1, 1, 1}};
prod1 = A1 . A2;
```

Now

```
MatrixForm[prod1]
```

results in the 3×3 matrix

```
5    -1   3

-5    4   -2

2    -1   1
```

Similarly, to determine the product A_2A_1 we enter

```
prod2 = A2 . A1;
MatrixForm[prod2]
```

which results in the 2×2 matrix

```
8    -5

0    2
```

Mathematica will return an error message if the dimensions of the matrices being multiplied are not compatible. For example,

```
Clear[a,b];
a={{1,2}, {3,4}};
b={1,2,3};
a . b
```

returns

```
Dot::dotsh:
   Tensors {{1, 2}, {3, 4}} and {1, 2, 3}
     have incompatible shapes.
```

Matrix and vector multiplication are carried out in the same manner. If c is the column vector

$$c = \begin{bmatrix} 1 \\ 3 \\ 9 \end{bmatrix},$$

then, after defining it as

```
Clear[c];
c = {1, 3 , 9}
```

we compute the product of B (defined earlier) and c by

```
B . c
```

which results in

```
{16, 3, 23}
```

Also the product of the column vector c^{T} with B is determined by

```
c . B
```

whose output is

```
{-8, 47, 11}
```

We note that the vector `c` could also have been defined as a 3×1 matrix:

```
Clear[c];
c={{1}, {3}, {9}};
```

Now `B . c` returns

```
{{16}, {3}, {23}}
```

while `c . B` returns the error message

```
Dot::dotsh:
   Tensors {{1}, {3}, {9}} and
    {{1, -1, 2}, {0, 1, 0}, {-1, 5, 1}}
     have incompatible shapes.
```

On the other hand, `Transpose[c] . B` returns

```
{{-8, 47, 11}}
```

There is a second way of multiplying lists in *Mathematica*, using the $*$ operand. This operation between two lists A and B of equal lengths returns a list whose entries are the product of individual entries of A and B. For example,

```
Clear[A, B];
A = {-1, 2}; B = {{1, 2,3 }, {3}};
A * B
```

returns

```
{{-1, -2, -3}, {6}}
```

Here both `A` and `B` have length 2. On the other hand, the product

```
Clear[A, B];
A = {{-1, 2}}; B = {{1, 2,3 }, {3}};
A * B
```

returns

```
Thread::tdlen:
   Objects of unequal length in
    {{-1, 2}} {{1, 2, 3}, {3}} cannot be combined.
```

The `Append` command appends information to the end of a list. For example,

```
B = {{1, 2, 3}, {-1, 0, 1}, {3, 4, 5}};
c = Append[B, {1, 1, 1}]
```

returns the 4×3 matrix with B as its first three rows and $\langle 1, 1, 1 \rangle$ as its fourth row. This command is particularly useful when, in the course of a calculation, new entries are being computed and this information must be added to a variable and stored for later processing.

The commands `Det`, `Inverse`, `Eigenvalues`, and `Eigenvectors` operate on lists, when applicable, with the standard mathematical results that their names suggest. For example, consider the matrix f

$$f(a) = \begin{bmatrix} 1 & a & 1 \\ -2 & 0 & a \\ -3a & 1 & a \end{bmatrix}.$$

Define this matrix in *Mathematica* by

```
f[a_] = {{1, a, 1}, {-2, 0, a}, {-3a, 1, a}}
```

Now

```
Det[f[a]]
```

returns

```
            2      3
-2 - a + 2 a  - 3 a
```

while

```
Inverse[f[a]]
```

leads to

```
                                        2
           a                      1 - a
{{-(--------------------), --------------------,
               2      3               2      3
    -2 - a + 2 a  - 3 a    -2 - a + 2 a  - 3 a

            2
           a
   --------------------},
              2      3
   -2 - a + 2 a  - 3 a

                2
       2 a - 3 a                4 a
 {--------------------, --------------------,
             2      3              2      3
  -2 - a + 2 a  - 3 a   -2 - a + 2 a  - 3 a

          -2 - a
  --------------------},
             2      3
  -2 - a + 2 a  - 3 a
```

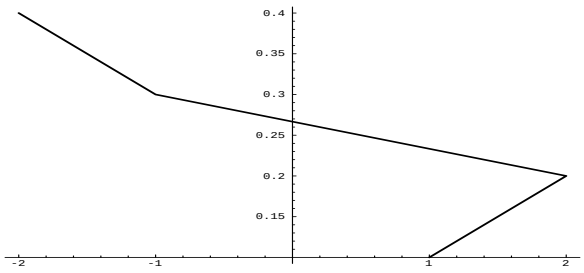

Figure 1.7: The output of `ListPlot`.

```
          2                   -1 - 3 a
{-(--------------------), --------------------,
               2      3                2      3
   -2 - a + 2 a  - 3 a    -2 - a + 2 a  - 3 a

       2 a
  --------------------}}
              2      3
  -2 - a + 2 a  - 3 a
```

Similarly, `Eigenvalues[f[1]]` and `Eigenvectors[f[2]]` return the appropriate outputs, respectively. The command `Eigensystem` combines the outputs of `Eigenvalues` and `Eigenvectors`.

We often need to plot a set of ordered pairs of numbers. The command `ListPlot` is the appropriate tool for this task. For example, consider the following four ordered pairs:

$$(1, 0.1), (2, 0.2), (-1, 0.3), (-2, 0.4).$$

To plot them, first we define a list that contains the four pairs and then apply `ListPlot` to it:

```
c = {{1, 0.1}, {2, 0.2}, {-1, 0.3}, {-2, 0.4}};
ListPlot[c, PlotJoined -> True]
```

The output is shown in Figure 1.7.

The above discussion touches on only a small portion of what is available in *Mathematica* concerning lists, matrices, and linear algebra. The reader is encouraged to consult [1] and [2] for more detail on this subject.

Figure 1.8: A mass-spring system.

1.10.1 Lists and Differential Equations

In certain problems where a large number of unknowns are present, we often end up writing down a system of ordinary differential equations that contains the information about the coupling between the variables. A typical example in engineering arises in the context of a set of masses and springs connected to each other in a series. The objective is to determine the oscillation of all masses in terms of the spring constants, the initial data, and any external forcing that may be present. We now give a simple example of such a system and outline how one proceeds to find the solution. We will use the concept of lists and present an algorithm that is nearly independent of the number of masses and springs present in the system.

Consider the mass–spring system in Figure 1.8.

Art Work ch1.1 (n masses, n+1 springs)

The governing system of differential equations is (see Chapter 7 in regards to the derivation of this system of equations)

$$\begin{aligned} m_1 x_1'' &= -k_1 x_1 + k_2(x_2 - x_1) \\ m_2 x_2'' &= -k_2(x_2 - x_1) + k_3(x_3 - x_2) \\ m_3 x_3'' &= -k_3(x_3 - x_2) + k_4(x_4 - x_3) \\ \ldots &= \\ \ldots &= \\ m_n x_n'' &= -k_n(x_n - x_{n-1}) + k_{n+1} x_n. \end{aligned}$$

In matrix notation, the above system is equivalent to

$$M\mathbf{x}'' = A\mathbf{x} \tag{1.11}$$

where

$$M = \begin{bmatrix} m_1 & 0 & 0 & \ldots & 0 \\ 0 & m_2 & 0 & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & \ldots & \ldots & 0 & m_n \end{bmatrix}$$

and

$$A = \begin{bmatrix} -(k_1 + k_2) & k_2 & 0 & \ldots & 0 \\ k_2 & -(k_2 + k_3) & k_3 & \ldots & 0 \\ 0 & k_3 & -(k_3 + k_4) & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & -(k_{n-1} + k_n) & k_n \\ 0 & 0 & \ldots & k_n & -k_n \end{bmatrix}.$$

System (1.11) is supplemented with the initial data

$$\mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}'(0) = \mathbf{x}_1. \tag{1.12}$$

Let us consider the concrete example of three unit masses attached to four identical springs with spring constants 10. The matrix M reduces to the identity matrix and A becomes

$$A = \begin{bmatrix} -20 & 10 & 0 \\ 10 & -20 & 10 \\ 0 & 10 & -20 \end{bmatrix}. \tag{1.13}$$

We consider the initial data

$$\mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}. \tag{1.14}$$

Our goal is solve (1.13) - (1.14) in *Mathematica* using `NDSolve`. We begin a new session of this program and introduce the number of variables `num` by

```
num = 3
```

and a list representing the solutions $x_i(t)$ by

```
vars = Table[x[i][t], {i, num}]
```

Next, we define the matrix A as

```
A = {{-20, 10, 0}, {10, -20, 10}, {0, 10, -20}}
```

We refer to the right-side of (1.11) by `rightside` and generate it by

```
rightside = A . vars
```

Next, we define the set of equations in (1.11) as the list

```
eqns = Table[x[i]''[t] == rightside[[i]], {i, num}]
```

The initial data (1.14) is stored in `initdata` directly:

```
initdata = {x[1][0] == 0, x[2][0] == 0, x[3][0] == 1,
              x[1]'[0] == 1, x[2]'[0] == -1, x[3]'[0] == 2}
```

Since `NDSolve` requires the differential equations and the initial data as one list, we use `Join` on `eqns` and `initdata`:

```
eqnsANDinit = Join[eqns, initdata]
```

We are now ready to apply `NDSolve`:

```
sol = NDSolve[eqnsANDinit, vars, {t, 0, 3}]
```

Mathematica responds with

```
{{x[1][t] -> InterpolatingFunction[{0., 3.}, <>][t],

     x[2][t] -> InterpolatingFunction[{0., 3.}, <>][t],

     x[3][t] -> InterpolatingFunction[{0., 3.}, <>][t]}}
```

To plot the three displacements on the same screen, we enter

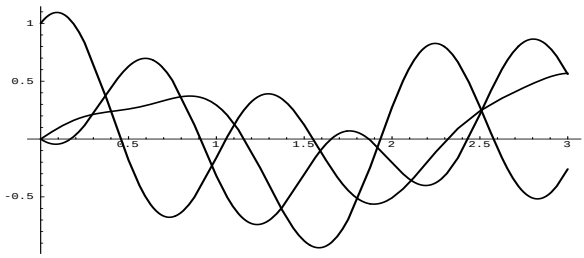

Figure 1.9: The output of the mass-spring system of differential equations.

```
Plot[Evaluate[Table[x[i][t] /. sol, {i, num}]], {t, 0, 3}]
```

Figure 1.9 shows the output. We have taken special care in implementing the various parts of (1.13) - (1.14) in such a way that only a few changes need be made in the above procedure when a different set of parameters is supplied to this system. This is particularly useful when *Mathematica* is being used in its Notebook setting, where the cut- and paste-mode offers a handy tool for introducing or altering parameters or definitions. Later in this chapter we will discuss how to write a program in *Mathematica* to reduce the number of changes one needs to make in order to analyze a problem such as (1.11) - (1.12) when the physical parameters are allowed to vary.

Problems

1. Let A and B be defined by

$$A = \begin{bmatrix} -1 & 1 \\ 1 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} -3 & 4 \\ 7 & 5 \end{bmatrix}.$$

   Compute $A + B$, $A - B$, AB, BA, $6A$, and $3A + 2B$.

2. Consider the matrix

$$A = \begin{bmatrix} a & b \\ b & c \end{bmatrix},$$

   where a, b, and c are constants. Find all values of these parameters for which the determinant of A vanishes.

3. Let A be the matrix

$$A = \begin{bmatrix} a & -a & b \\ -a & b & a \\ b & a & 2a \end{bmatrix}.$$

Find all values of a and b for which the determinant of A vanishes.

4. Let A be the matrix
$$A = \begin{bmatrix} a & b \\ b & a \end{bmatrix}.$$
   (a) Compute A^5 and A^{10} (use `MatrixPower`).
   (b) Use `Exp` and `MatrixExp` with A. Why are the results different?

5. Consider the matrix
$$A = \begin{bmatrix} -1 & 1 \\ 1 & 3 \end{bmatrix}.$$
   Let $s(n)$ be the nth partial sum of the Taylor series expansion of e^A, that is,
$$s(n) = I + \sum_{i=1}^{n} \frac{1}{i!} A^i,$$
   where I is the 2×2 identity matrix. Find $s(2)$, $s(5)$, and $s(10)$. Compare these matrices to the output of `MatrixExp[A]`.

6. Start a session in *Mathematica* and generate Figure 1.9.

7. Generate the analogue of Figure 1.9 in the following settings. All masses are assumed to be one.
   (a) $k_1 = k_2 = 10$, $k_3 = 20$; $x_1(0) = 0, x_2(0) = -1, x_3(0) = 1$, $x_1'(0) = 0, x_2'(0) = 0, x_3'(0) = 0$; $t \in (0, 3)$
   (b) $k_1 = k_2 = k_3 = k_4 = 10$; $x_1(0) = 0, x_2(0) = 0, x_3(0) = 0$, $x_4(0) = 1$, $x_1'(0) = 0$, $x_2'(0) = 0$, $x_3'(0) = 0$, $x_4'(0) = 0$; $t \in (0, 5)$

8. Consider the mass–spring system in Figure 1.8 containing five equal masses with $m_i = 5$, $i = 1, ..., 5$, and six springs with spring constants $k_i = 3$, $i = 1, ..., 6$. While all other masses are kept stationary, we perturb this system from its equilibrium by displacing the third mass one unit to its right and releasing it. Find the displacement of all of the masses for $t \in (0, 10)$.

9. Consider the mass–spring system in Figure 1.8 consisting of three unequal masses $m_1 = 1$, $m_2 = 3$, and $m_3 = 5$. Four identical springs with spring constants $k = 20$ are attached to these masses. Assuming that the initial disturbance in this system is caused by displacing the first mass one unit to its left and then releasing it, generate the analogue of Figure 1.9 for $t \in (0, 10)$.

10. Consider the mass–spring system consisting of two identical masses $m_1 = m_2 = 1$, attached to three identical springs with spring constant k. Let $x_1(0) = 1$, $x_2(0) = 0$, and $x_1'(0) = x_2'(0) = 0$. Let $x_i(t, k)$ denote the displacement of the ith mass at time t. Use `NDSolve` to find the value for $x_1(3, k)$ as k takes on values from 1 to 10 at increments of 0.1. Let S be the list containing the ordered pairs $(k, x_1(3, k))$. Use `ListPlot` and graph S. Compare your graph with Figure 1.10. (Hint: One way to get Figure 1.10 is make the following changes to the process discussed in Section 1.10.1. Besides the necessary modifications to prepare for the above two-dimensional system, define the matrix `A` in terms of `k`, that is

```
A = {{-2 k, k}, {k, -2k}};
```

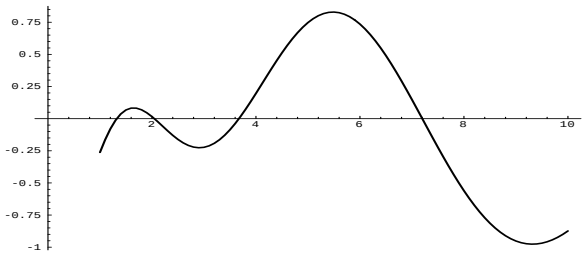

Figure 1.10: The graph of $x_1(3, k)$.

Next, enhance the definition of `sol` to allow for the dependence on the spring constant:

```
sol[j_] := NDSolve[eqnsANDinit/. k -> j, vars, {t, 0, 3}];
```

(the significance of using `:=` as opposed to `=` is discussed in the next section). Finally, generate the points for the graph by

```
points = Table[{k, First[x[1][t] /. sol[k]/. t->3]}, {k,1,10,0.1}];
```

)

1.11 The =, :=, ==, ->, /. Operators

The operation `=` has the same standard definition in *Mathematica* as in many other computer languages. The expression

```
a = b
```

means that the value stored in `b` replaces the value stored in `a` and becomes the current value of that symbol. In particular, `=` should not be used when defining an equation, as we have seen in the discussion of `Solve` and `DSolve`, among other internal functions of *Mathematica*. Also, the expression

```
t = t + h
```

does not imply that $h = 0$; it simply states that the old value of `t` is added to the current of `h`, and the result becomes the current (new) value of `t`. This expression is the common way of updating a variable in a recursive process, as we will see in the next section.

The operator := denotes delayed evaluation and is most useful in definitions where we are not yet ready to use an assignment. Compare the output of

```
t = 3;
f[t_] = Sin[t];
f[4]
```

with the output of

```
t = 3;
g[t_] := Sin[t];
g[4]
```

In the first case we see that `Sin[3]` is the output, since the value of `t` is passed to the argument of `f` as soon as its definition is executed. In the second case, where the output is `Sin[4]`, the function `Sin[t]` is assigned to `g`, but nothing is executed until the next time `g` is called upon.

We have already encountered == on many occasions. This operation performs the task of testing for equality and defining equations. For instance,

```
x == y
```

tests to see whether `x` and `y` are equal to each other. Try

```
Clear[x,y];
x = 4; y = 5; x == y
```

== is also used in defining equations as in

```
DSolve[x''[t] + x'[t] == 0, x[t], t]
```

The operations `/.` and `->` are used together for the purpose of replacement by the rule(s) defined by `->`. For example,

```
Clear[x];
x^2 - 4*x + 3 /. x -> -2
```

results in the evaluation of the quadratic polynomial at $x = 2$. Similarly,

```
Integrate[f[t], {t, 0, 1}] /. f[t_] -> Exp[t]
```

computes $\int_0^1 e^t \, dt$. It is also possible to use `->` with multiple rule assignments:

```
Clear[a,x,b,y];
a x^2 + b y^2 /. {a->1, b->-3, x->2, y->-1}
```

or

```
Clear[a,x,b,y,c,d,e,f];
Solve[{a x + b y == c, d x + e y == f/. a -> 1/. c -> -1}]
```

Problems

1. Determine the outcome of the following statements.

   (a) `t == 3`

   (b) `t = 3; f[t_] = Sin[t] Cos[t]; f[Pi]`

   (c) `t = 3; f[t_] := Sin[t] Cos[t]; f[Pi]`

   (d) `Sin[a t + b] /.  a -> Pi /.  b -> 0 /.  t -> 1/2`

   (e) `Solve[x^2 - a == 0 /.  a -> 3, x]`

   (f) `Solve[x^2 - a == 0, x ] /.  a -> 3`

   (g) `D[Sin[t] /.  t -> 3, t]`

   (h) `D[sin[t], t] /.  t -> 3`

2. Find the syntax errors in the following statements.

   (a) `Dsolve[x^2 - 4 = 0, x]`

   (b) `Integrate[f[t] = t^2, {t, 0, 1}]`

   (c) `D[Sin[t], t /.  t -> 3]`

   (d) `DSolve[x'[t] + x[t] == -1, x[0] = 2, x[t], t]`

1.12 Loops and the `Do` Command

In many numerical applications we need to perform an operation repeatedly while a few parameters may change with each iteration. The `Do` command in *Mathematica* is the right tool for such a task. As a first example, consider the sum

$$S = \sum_{i=1}^{100} \frac{1}{i^2}.$$

One can find an approximate value for S using `Do` via

```
S = 1.;
Do[S = S + 1/i^2, {i, 2, 100}];
S
```

Mathematica returns `1.63498`. (Try the last program with `S = 1` replacing the first line. How are the outputs different?) We also get the same result from `Sum`:

```
Sum[N[1/i^2], {i, 1, 100}]
```

A different context in which `Do` is useful is in carrying out the type of iterations that arise naturally in the discretization of differential equations. A simple example of this type of application appears in the numerical scheme that produces the Euler approximation of the solution of a first-order differential equation. Consider the first-order differential equation

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0.$$

The Euler approximation of the solution $y(x)$ seeks a sequence (x_n, y_n) that satisfies the difference equation

$$y_{n+1} = y_n + hf(x_n, y_n),$$

with $x_{n+1} = x_n + h$, where h is a fixed small positive number. The following program shows how to generate this sequence and plot the approximate solution using `ListPlot`. This program is written for $f(x, y) = -y + \sin x$, $x_0 = 0$, $y_0 = 1$, $h = 0.01$, and $n = 10$.

```
Clear[f, x, y];
f[x_, y_] = -y + Sin[x];
x0 = 0; y0 = 1;
h = 0.01;
n = 10;
x = x0; y = y0;
output = {{x, y}};
Do[y = y + h*f[x, y]; x = x + h;
     output = Append[output, {x, y}], {i, 1, n}];
ListPlot[output, PlotJoined -> True]
```

The function of `Do` in the above program is to perform `y = y + h*f[x, y]` repeatedly, while updating the value of `x` and appending the result of `x` and `y` to `output`.

`Do` is also quite useful in finding fixed points of functions: A fixed point of a function f is a point x such that $f(x) = x$. For example, $x = 0$ is a fixed point of the sine function. One way to approximate a fixed point of a function is to generate the sequence

$$x_0 = \text{initial guess}, \quad x_1 = f(x_0), \quad x_2 = f(x_1),$$

and in general

$$x_n = f(x_{n-1}).$$

If $\{x_n\}$ converges to a, and if f is a continuous function, then a is a fixed point of f. This algorithm is rather easy to implement using `Do` (the following program is written for $f(x) = 2 \sin x$, with $x_0 = \frac{1}{2}$):

```
Clear[f, x];
f[x_] = 2 Sin[x];
x = 0.5;
Do[y = N[f[x]]; Print[{x, y}]; x = y, {i, 20}]
```

Its output is

```
{0.5, 0.958851}
{0.958851, 1.63706}
{1.63706, 1.99561}
{1.99561, 1.82223}
{1.82223, 1.93711}
{1.93711, 1.86731}
{1.86731, 1.91272}
{1.91272, 1.88422}
{1.88422, 1.90257}
{1.90257, 1.89093}
{1.89093, 1.89838}
{1.89838, 1.89364}
{1.89364, 1.89667}
{1.89667, 1.89474}
```

```
{1.89474, 1.89597}
{1.89597, 1.89519}
{1.89519, 1.89569}
{1.89569, 1.89537}
{1.89537, 1.89557}
{1.89557, 1.89544}
```

Problems

1. Use `Do` and sum the following series.
   (a) $\sum_{i=0}^{20} i$
   (b) $\sum_{i=1}^{10} \frac{1}{i^2}$
   (c) $\sum_{i=1}^{100} \frac{1}{i^2}$. First sum the series using exact arithmetic and then using floating point arithmetic (that is, use the decimal representation of $\frac{1}{i^2}$).
   (d) $\sum_{i=1}^{1000} \frac{1}{i}$. What is the exact value of the sum? Find its forty decimal point approximation (use `N[number, 40]`). Compare this value with the value of the sum when the decimal representation of $\frac{1}{i}$ is used.

2. Find the fixed points of the following functions. Experiment with the number of iterations to get a sense of whether the iteration process converges or not.
   (a) $f(x) = \sin 2x; \quad x_0 = \frac{1}{2}$
   (b) $f(x) = \sin 2x; \quad x_0 = \frac{3}{2}$
   (c) $f(x) = -\sin 2x; \quad x_0 = \frac{1}{2}$
   (d) $f(x) = \sqrt{x} + 1; \quad x_0 = 1$
   (e) $f(x) = \sin \sqrt{x} + 1; \quad x_0 = 1$
   (f) $f(x) = \frac{1}{x^2+1}; \quad x_0 = 1$. Compare the result with the output of

   ```
   Clear[x];
   FindRoot[1/(x^2+1) == x, {x, 1}]
   ```

   (g) $f(x) = \ln 2x; \quad x_0 = \frac{1}{2}$

3. Use `Do` to determine the first ten iterations of

$$f_1(x) = \sin x, \quad f_i(x) = \int_0^x e^{-t} f_{i-1}(t)\, dt.$$

1.13 Examples of Programming in *Mathematica*

On several occasions in this chapter we have encountered situations where the process of answering questions in a problem set requires repeated execution of a series of instructions in *Mathematica* in which only a few parameters change. With the Notebook interface of this software we could reduce the amount of typing by cutting and pasting input lines already present in a session. We

now introduce an alternative method that is quite efficient and works especially well with projects that require a large number of input statements.

A useful feature of *Mathematica* is that it allows one to input lines of commands from an external file. Using this feature and combining a series of internal functions, we can construct new functions that are specifically customized for certain objectives. We give an example of such a "program" in the context of differential equations. Its task is to solve a system of differential equations and plot the solution to an initial value problem.

Let us consider the system of differential equations

$$\frac{dx}{dt} = f(x, y, t), \quad \frac{dy}{dt} = g(x, y, t) \tag{1.15}$$

subject to the initial data

$$x(0) = x0, \quad y(0) = y0, \tag{1.16}$$

where

$$f(x, y, t) = x - y + \sin t, \quad g(x, y, t) = x + y + \cos t \tag{1.17}$$

and

$$x0 = 0.1, \quad y0 = 1.2. \tag{1.18}$$

We wish to plot the solution of this system over the interval $(0, 3)$. The following lines are saved in a file called `ode.m`:

```
Clear[x,y,tfinal,sol];
f[x_, y_, t_] = x - y + Sin[t];
g[x_, y_, t_] = x + y + Cos[t];
x0 = 0.1; y0 = 1.2; tfinal = 3;
sol = NDSolve[{x'[t] == f[x[t], y[t], t],
               y'[t] == g[x[t], y[t], t],
               x[0] == x0, y[0] == y0},
               {x, y}, {t, 0, tfinal}];
output = ParametricPlot[Evaluate[{x[t], y[t]} /. sol], {t, 0, tfinal}]
```

Remark 1.13.1: When using a word processing software (such as *WordPerfect*) to create files for use in *Mathematica*, it is a good habit to save the files as text-only.

Then, after initiating *Mathematica*, we input `ode.m` by entering

```
<<ode.m
```

Clearly, if we intend to solve a different set of differential equations we only need to alter the lines in `ode.m` that define `f` and `g` and input the new `ode.m` program to *Mathematica*.

An alternative to re-editing the file `ode.m` for each set of differential equations is to construct a function from the block of instructions in `ode.m`, whose input would be the functions f and g (and other parameters, if we wish), and whose output is the graph of the solution through the initial data. The `Block` command is the appropriate tool for this task. Additionally, we are often interested in the phase plane diagram of the system (1.15), that is, a graph that contains plots of solutions of this system through several typical initial data. We will use the internal function `Flatten` for this purpose. This function acts on lists and flattens out nested lists. For example,

```
Flatten[{{1,2}, {3,4}}]
```

returns

```
{1, 2, 3, 4}
```

Also, `Flatten[list, n]` flattens `list` to level `n`. With these remarks in mind, the following is the listing of the program `odesolver.m`. It takes the parameters

```
f, g, data, tfinal
```

and delivers the phase diagram of (1.15) as output:

```
odesolver[f_, g_, data_, tfinal_] := Block[{x, y},
   solution=Table[NDSolve[{x'[t]==f[x[t],y[t],t],
                   y'[t]==g[x[t],y[t],t],
                   x[0]==data[[i]][[1]],
                   y[0]==data[[i]][[2]]},
                   {x,y},{t,0,tfinal}],
       {i,1,Length[data]}];
   solution=Flatten[solution,1];
   OutPut=ParametricPlot[Evaluate[{x[t],y[t]}/.solution],{t,0,tfinal}];
   Print["The graph is stored in OutPut"];
]
```

To illustrate the utility of `odesolver.m`, we now plot the graphs of approximate solutions to the system

$$x' = y, \quad y' = -y + \sin x,$$

which pass through the following three initial points

$$(1,1), \quad (2,2), \quad \text{and } (3,3).$$

Start a new session of *Mathematica* and enter

```
<< odesolver.m
```

At the prompt, enter the right-side of the system of differential equations

```
f[x_, y_, t_] = y; g[x_, y_, t_] = -y - Sin[x];
```

and the initial data and `tfinal`

```
tfinal = 10; data = {{1,1}, {2,2}, {3,3}};
```

Next, enter

```
odesolver[f, g, data, tfinal]
```

Mathematica now goes to work, plots the graph of the three trajectories, and returns

```
The graph is stored in OutPut
```

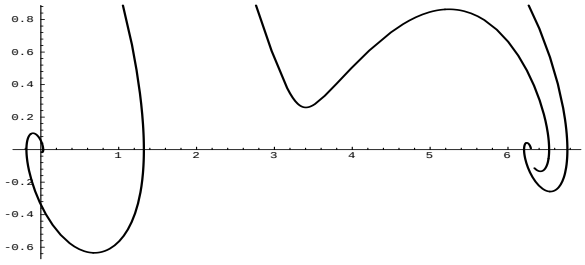

Figure 1.11: The output of `odesolver.m` for the system $x' = y, y' = -y - \sin x$.

(see Figure 1.11). The symbol `OutPut` now contains all of the information about the phase plane. We could apply `Show`, `PSPrint`, or `AspectRatio` to `OutPut`, if we wish. It is also possible to make the entire process of entering the functions f and g and the initial data an interactive process. The `Input` command in *Mathematica* is quite useful in these circumstances. The following program, which we call `myode.m`, has this interactive feature built into it.

```
Block[{x,y,f,g,data},
   f[x_, y_, t_] = Input["This program integrates x' = f(x,y,t), y' = g(x,y,t)
   \n
   Enter f (for example x + y + Sin[t])
   \n"];
   g[x_,y_,t_]=Input["\n
   Input g (for example x - y - Cos[t])
   \n"];
   data=Input["\n
   Enter the initial data (for example {{0, 1}, {-1, 2}}
   \n"];
   tfinal=Input["\n
   Enter the length of integration in t (for example, 3)
   \n"];
   sol=Table[NDSolve[{x'[t]==f[x[t],y[t],t],
                   y'[t]==g[x[t],y[t],t],
                   x[0]==data[[i]][[1]],
                   y[0]==data[[i]][[2]]},
                   {x,y},{t,0,tfinal}],
       {i,1,Length[data]}];
   solution=Flatten[sol,1];
```

```
   OutPut=ParametricPlot[Evaluate[{x[t],y[t]}/.solution],{t,0,tfinal}];
   Print["The graph is stored in OutPut"];
]
```

Let us apply `myode.m` to

$$x' = \frac{\partial H}{\partial y}, \quad y' = -\frac{\partial H}{\partial x}. \tag{1.19}$$

Such a system is often called a Hamiltonian system of differential equations, and H is called a hamiltonian or a stream function of the system. Such systems are prevalent in fluid dynamics, where the ordered pair $(x(t), y(t))$ represents the position of a fluid particle at time t. Among them is the system of equations of motions of particles of fluid that pass a cylinder, where H is explicitly known to be

$$H = y - \frac{y}{x^2 + y^2}.$$

We now use `myode.m` and plot the particle paths of the fluid particles that are initially located at

$$(-3, 0.1i), \quad i = 1, ..., 5.$$

Start a new session of *Mathematica*. First, define H by

```
H = y - y/(x^2 + y^2)
```

Next, call up `myode.m`:

```
<<myode.m
```

When prompted for `f` and `g` enter

```
D[H, y]
```

and

```
- D[H, x]
```

respectively, Next, enter the initial data

```
Table[{-3, 0.1 i}, {i, 1, 5}]
```

Finally, choose `tfinal` to be 5. Figure 1.12 shows the output that is stored in `OutPut`.

Problems

1. Use an editor and create the files `ode.m`, `odesolver.m`, and `myode.m`. Study the logic of each program carefully. Run these programs separately in *Mathematica*, and generate Figures 1.11 and 1.12.

2. Use `ode.m` to plot the trajectories of the following systems of differential equations.

   (a) $x' = y, \quad y' = -x$; $x(0) = 2$, $y(0) = -3$; $0 < t < 4$

   (b) $x' = 2x - y, \quad y' = x + y$; $x(0) = -1$, $y(0) = 1$; $0 < t < 3$

   (c) $x' = \frac{y}{\sqrt{x^2+y^2}}, \quad y' = -\frac{x}{\sqrt{x^2+y^2}}$; $x(0) = -2$, $y(0) = 2$; $0 < t < 4$

   (d) $x' = e^{-t}y, \quad y' = -x + e^{-3t}$; $x(0) = 1$, $y(0) = -1$; $0 < t < 1$.

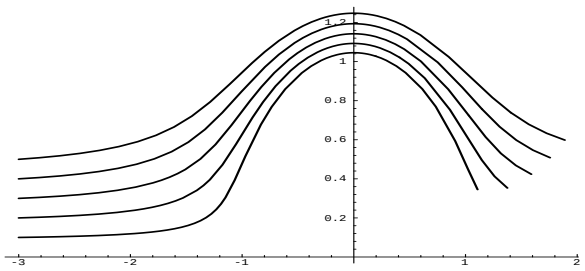

Figure 1.12: The output of `myode.m` for the flow past cylinder.

3. Execute `odesolver.m` or `myode.m` to generate the phase plane diagram for each of the following systems. In each experiment with the initial data and the value of `tfinal`.

   (a) $x' = y$, $y' = -0.1y - \sin y$

   (b) $x' = y$, $y' = -y - y^3$

   (c) $x' = x + y(1 - x^2 - y^2)$, $y' = y - x(1 - x^2 - y^2)$

1.14 Glossary of Useful Commands

1. **PSPrint**

   This command is used primarily in the stand-alone version of *Mathematica*. The entry `PSPrint[Out[x]]`, where `x` marks the output you wish to print, creates a postscript hardcopy of the graphics created and stored in `Out[x]`. In the Notebook interface version of *Mathematica*, the `Print` command in the menu performs this task.

2. **Table**

   The `Table` command creates a list of objects. Its syntax is

   ```
   Table[expr, {i, n}, {j, m}]
   ```

   where a list of copies of `expr` are generated as `i` and `j` vary from 1 to `n` and `m`, respectively. For example,

   ```
   Table[i*j, {i, 1, 3}, {j, 1, 4}]
   ```

returns

```
{{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 6, 9, 12}}
```

3. **Plot, Plot3D, ParametricPlot**, and **ParametricPlot3D**

   The commands `Plot`, `Plot3D`, `ParametricPlot`, and `ParametricPlot3D` draw graphs of various two- and three-dimensional representations of functions. For example,

   ```
   Plot[Sin[x], {x, 0, 2 Pi}]
   ```

   draws the graph of $f(x) = \sin x$ over the interval $(0, 2\pi)$, while

   ```
   ParametricPlot[{t, Sin[t]}, {t, 0, 2 Pi}]
   ```

   accomplishes the same task using a typical parametrization of the same curve. The commands `Plot3D` and `ParametricPlot3D` have similar syntax:

   ```
   Plot3D[Sin[x] Cos[y], {x, 0, 6 Pi}, {y, 0, 6 Pi}];
   ```

   and

   ```
   ParametricPlot3D[{x, y, Sin[x] Cos[y]}, {x, 0, 6 Pi}, {y, 0, 6 Pi}]
   ```

   yield the same results.

4. **Solve** and **FindRoot**

   `Solve` and `FindRoot` find solutions of equations. Typical examples are

   ```
   Solve[x^2 + 2 x + a == 0, x]
   ```

   and

   ```
   FindRoot[x^2 + 2 x + 1 == 0, {x, 3}]
   ```

5. **DSolve** and **NDSolve**

   `DSolve` and `NDSolve` find solutions to ordinary differential equations. Their syntax follows the pattern

   ```
   DSolve[equations, dependent variables, independent variable]
   ```

   and

   ```
   NDSolve[{equations, initial data}, dependent variables, independent variable]
   ```

   For example,

   ```
   DSolve[{x''[t] + 2 x[t] == 0, x[0] == 1, x'[0] == 2}, x[t], t]
   ```

   and

```
NDSolve[{x''[t] + 2 Sin[x[t]] == 0, x[0] == 1, x'[0] == 2}, x[t], {t, 0, 3}]
```

6. **LaplaceTransform** and **InverseLaplaceTransform**

   The package that enables *Mathematica* to compute the Laplace transform of a function is `Calculus‘LaplaceTransform‘`. It should be entered in *Mathematica* at the beginning of a session as

   ```
   <<Calculus‘LaplaceTransform‘
   ```

   Typical commands for computing the Laplace transform and inverse transforms of functions are

   ```
   LaplaceTransform[t Sin[t], t, s]
   ```

   and

   ```
   InverseLaplaceTransform[(s-3)/(s^2 + 2s +9), s, t]
   ```

References

[1] Wolfram, Stephen, *Mathematica*, 3rd Edition, Addison - Wesley Publishing Co., Reading, MA, 1996.

[2] Blachman, Nancy, *Mathematica: A Practical Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1992.