C. Woodford
C. Phillips

Numerical Methods with Worked Examples: Matlab Edition

Second Edition

Springer

Numerical Methods with Worked Examples: Matlab Edition

C. Woodford • C. Phillips

Numerical Methods with Worked Examples: Matlab Edition

Second Edition

Springer

C. Woodford
Department of Computing Service
Newcastle University
Newcastle upon Tyne, NE1 7RU
UK
c.h.woodford@newcastle.ac.uk

Prof. C. Phillips
School of Computing Science
Newcastle University
Newcastle upon Tyne, NE1 7RU
UK

Additional material to this book can be downloaded from http://extras.springer.com.

ISBN 978-94-007-1365-9 e-ISBN 978-94-007-1366-6
DOI 10.1007/978-94-007-1366-6
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2011937948

Cover design: deblik

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This book is a survey of the numerical methods that are common to undergraduate courses in Science, Computing, Engineering and Technology. The aim is to present sufficient methods to facilitate the numerical analysis of mathematical models likely to be encountered in practice. Examples of such models include the linear equations describing the stress on girders, bridges and other civil engineering structures, the differential equations of chemical and thermal reactions, and the inferences to be drawn from observed data.

The book is written primarily for the student, experimental scientist and design engineer for whom it should provide a range of basic tools. The presentation is novel in that mathematical justification follows rather than precedes the description of any method. We encourage the reader first to gain a familiarity with a particular method through experiment. This is the approach we use when teaching this material in university courses. We feel it is a necessary precursor to understanding the underlying mathematics. The aim at all times is to use the experience of numerical experiment and a feel for the mathematics to apply numerical methods efficiently and effectively.

Methods are presented in a *problem–solution–discussion* order. The *solution* may not be the most elegant but it represents the one most likely to suggest itself on the basis of preceding material. The ensuing *discussion* may well point the way to better things. Dwelling on practical issues we have avoided traditional problems having neat, analytical solutions in favour of those drawn from more realistic modelling situations which generally have no analytic solution.

It is accepted that the best way to learn is to teach. But even more so, the best way to understand a mathematical procedure is to implement the method on a totally unforgiving computer. Matlab enables mathematics as it is written on paper to be transferred to a computer with unrivalled ease and so offers every encouragement. The book will show how programs for a wide range of problems from solving equations to finding optimum solutions may be developed. However we are not recommending re-inventing the wheel. Matlab provides an enormous range of ready to use programs. Our aim is to give insight into which programs to use, what may be expected and how results are to be interpreted. To this end we will include details of the Matlab versions of the programs we develop and how they are to be employed.

We hope that readers will enjoy our book. It has been a refreshing experience to reverse the usual form of presentation. We have tried to simplify the mathematics as far as possible, and to use inference and experience rather than formal proof as a first step towards a deeper understanding. Numerical analysis is as much an art as a science and like its best practitioners we should be prepared to pick and choose from the methods at our disposal to solve the problem at hand. Experience, a readiness to experiment and not least a healthy scepticism when examining computer output are qualities to be encouraged.

Newcastle University, Newcastle upon Tyne, UK

Chris Woodford
Chris Phillips

Contents

Chapter 1
Basic Matlab

Aims This introductory chapter aims to encourage the immediate use of Matlab through examples and exercises of basic procedures. Sufficient techniques and skills will be illustrated and practised to prepare for the end of chapter exercises in the main text and indeed for any numerical calculation. Presentation of results and reading and writing data from files will also be considered. Overall it is hoped that the beauty, simplicity and power of Matlab will begin to appear.

1.1 Matlab—The History and the Product

Matlab was devised by Cleve Moler in the late 1970's to make numerical computing easier for students at the University of New Mexico. Many of the obstacles to using a computer for mathematics were removed. In a Matlab program variables, whether real or complex numbers, vectors or matrices may be named and used as and when required without prior notification or declaration and may be manipulated according to the rules of mathematics. Matlab spread to other Universities and in 1984 Cleve went into partnership with a colleague to set up a company called Mathworks to market Matlab. Mathworks is now a multi-national corporation specialising in technical computing software. Matlab and products built on Matlab are used all over the world by innovative technology companies, government research labs, financial institutions, and more than 3,500 universities.

Matlab is a high-level user friendly technical computing language and interactive environment for algorithm development, data visualisation, data analysis, and numeric computation. Matlab applications include signal and image processing, communications, control design, test and measurement, financial modelling and analysis, and computational biology. Matlab provides specialised collections of programs applicable to particular problem areas in what are known as Toolboxes and which represent the collaborative efforts of top researchers from all over the world. Matlab can interface with other languages and applications and is the foundation for all MathWorks products. Matlab has an active user community that contributes freely available Matlab programs to a supported web site[1] for distribution and appraisal. Help, advice and discussion of matters of common interest are always available in internet user forums. Whatever the question, someone somewhere in the world will have the answer.

1.2 Creating Variables and Using Basic Arithmetic

To create a single variable just use it on the left hand side of an equal sign. Enter the name of the variable to see its current value.

The ordinary rules of arithmetic apply using the symbols $+$, $-$, * and \ for addition, subtraction, multiplication and division and $\wedge$ raising to a power. Use round brackets (and) to avoid any ambiguities.

In the interests of clarity when illustrating Matlab commands the $\wedge$ symbol will be used throughout the book to denote the caret (or circumflex) entered from most keyboards by the combination SHIFT-6.

Exercises

1. Enter the following commands[2] on separate lines in the command window.

(i)	r = 4	(ii)	ab1 = 7.1 + 3.2	(iii)	r
(iv)	A = r∧2	(v)	sol = ab1∗(1 + 1/r)	(vi)	ab1 = sol∧ (1/2)
(vii)	CV2 = 10/3	(viiii)	x = 1.5e-2		

2. Enter one or more of the previous commands on the same line using a semicolon ; to terminate each command. Notice how the semi-colon may be used to suppress output.
3. If £1000 is owed on a credit card and the APR is 17.5% how much interest would be due? Divide the APR by 12 to get the monthly rate.

1.3 Standard Functions

Matlab provides a large number of commonly used functions including *abs*, *sqrt*, *exp*, *log* and *sin*, *cos* and *tan* and inverses *asin*, *acos*, *atan*. Functions are called using brackets to hold the argument, for example

```
x = sqrt(2);     A = sin(1.5);     a = sqrt(a∧2 + b∧2);     y = exp(1/x);
```

For a full list of Matlab provided functions with definitions and examples, follow the links *Help* → *Product Help* from the command window and choose either *Functions By Category* or *Functions Alphabetical List* from the ensuing window.

[1] www.mathworks.com/matlabcentral/fileexchange/

[2] We may also terms such as *statement*, *expression* and *code* within the context of Matlab commands.

Exercises

1. Find the longest side of a right angled triangle whose other sides have lengths 12 and 5. Use Pythagoras: $(a^2+b^2=c^2)$.
2. Find the roots of the quadratic equation $3x^2-13x+4$ using the formula $x=(-b\pm\sqrt{b^2-4ac})/2a$. Calculate each root separately. Note that arithmetic expressions with an implied multiplication such as $2a$ are entered as 2∗a in Matlab.
3. Given a triangle with angle $\pi/6$ between two sides of lengths 5 and 7, use the cosine rule $(c^2=a^2+b^2-2ab\cos C)$ to find the third side.

Note that π is available in Matlab as pi.

1.4 Vectors and Matrices

Vectors and matrices are defined using square brackets. The semi-colon ; is used within the square brackets to separate row values.

For example the matrix

$$\mathbf{A}=\begin{pmatrix}2&3&-1\\4&8&-3\\-2&3&1\end{pmatrix}$$

may be created using

A = [2 3 −1; 4 8 −3; −2 3 1]

A row vector $\mathbf{v}=(2\ \ 3\ \ -1)$ may be created using v = [2 3 −1].

The transpose of a vector (or matrix) may be formed using the ' notation. It follows that the column vector

$$\mathbf{w}=\begin{pmatrix}5\\3\\2\end{pmatrix}$$

may be created using either w = [5; 3; 2] or w = [5 3 2]'.

Exercises

1. Create the row vector $x=(4\ \ 10\ \ -1\ \ 0)$.
2. Create the column vector $y=(-5.3\ \ -2\ \ 0.9\ \ 1)$.
3. Create the matrix

$$\mathbf{B}=\begin{pmatrix}1&7.3&-5.6&2\\1.4&8&-3&0\\-2&6.3&1&-2\end{pmatrix}.$$

4. The roots of a polynomial may be found by using the function *roots*. For example the command *roots*(p) will return the roots of a polynomial, where p

specifies the vector of coefficients of the polynomial beginning with the coefficient of the highest power and descending to the constant term. For example *roots*([1 2 −3]) would find the roots of $x^2 + 2x - 3$. Use *roots* to check the solution of question (2), Sect. 1.3.

Individual elements of a vector or matrix may be selected using row-column indices, for example by commands such as A(1, 2) or z = w(3).

Whole rows (or columns) may be selected. For example, A(: , 1) selects the 1st column and A(3 , :) selects the 3rd row of a matrix **A**. More generally, having assigned values to variables i and j, use expressions such as *abc* = A(: , j) to assign the jth column of **A** to a variable, *abc* and *xyz* = A (i , :) to assign the ith row of **A** to a variable, *xyz*.

Exercises

1. Using **B** (as above) assign the values held at **B**(1, 2) to a variable, a and **B**(3, 4) to variable, b.
2. Form a vector, v1 from the 2nd column of **B** and the vector, v2 from the 1st row of **B**.

Note that Matlab enforces the rules regarding the multiplication of matrices and vectors. A command of the form A∗B is only executed if **A** has the same number of columns as **B** has rows.

Exercises

1. Using the previous A and w form the product A∗w. See what happens if the order of multiplication is reversed.
2. The norm (or *size*) of a vector, $\mathbf{v}$ where $\mathbf{v} = (v_1, \ldots, v_n)$ is defined to be $\sqrt{v_1^2 + \cdots + v_n^2}$. Find the norm of the vector (3 4 5 −1) either by adding individual terms and taking the square root or by using the formula $\sqrt{v.v^T}$, where v is a row vector. Check the result using the Matlab function *norm* and a command of the form *norm*(v).

In general inserting spaces in Matlab commands does not affect the outcome. However care has to be taken when forming a list of numbers as is the case when specifying the elements of a vector or an array. The notation for numbers takes priority over the notation for expressions connecting numbers with operators. In practice this means that the command, V = [1 2 −1] produces the 3-element vector, [1 2 −1]. On the other hand, V = [1 2 – 1] produces the 2-element vector, [1 1].

In addition to the usual operations for adding, subtracting and multiplication of compatible matrices and vectors Matlab provides a large number of other matrix–vector operations. Use the Matlab help system and follow the links *Help* → *Product Help* → *Functions: By Category* → *Mathematics* → *Arrays and Matrices* (and also → *Linear Algebra*) for details. As example we consider the left-division \ operator.

A system of linear equations such as

$$\begin{aligned} 3x_1 + 5x_2 + 7x_3 &= 25, \\ x_1 - 4x_2 + 2x_3 &= 10, \\ 4x_1 - x_2 - 3x_3 &= -1 \end{aligned}$$

may be written in the form $\mathbf{Ax} = \mathbf{b}$, where

$$\mathbf{A} = \begin{pmatrix} 3 & 5 & 7 \\ 1 & -4 & 2 \\ 4 & -1 & -3 \end{pmatrix}, \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 25 \\ 10 \\ -1 \end{pmatrix}.$$

If A is an $n \times n$ matrix and b is a column vector having n rows then the solution (if it exists) to the linear system of equations written in the form $\mathbf{Ax} = \mathbf{b}$ is given in Matlab by using the command

```
x = A\b
```

A warning is given if the solution does not exist or is unreliable due to rounding error.

Exercises

1. Solve the following systems of equations.

$$\text{(i)}\ \begin{aligned} 2\,a + \,c &= 5 \\ 3\,a - 2\,c &= 4 \end{aligned} \qquad \text{(ii)}\ \begin{aligned} 3\,a + 2\,c - \,d &= 3 \\ 2\,a - 2\,c + 3\,d &= 9 \\ a - \,c - \,d &= 2. \end{aligned}$$

1.5 M-Files

On the whole entering Matlab commands and statements line by line is too error prone for anything but very short, transitory programs. Programs under development and programs which are going to be used repeatedly are better stored as a sequence of commands in a file. Such files are called M-files. A single command, *filename* if that is the name of the M-file stored in the working directory, causes Matlab to execute the commands stored in *filename*. An M-file is created by opening the *File* menu in the command window and following the link *New* → M-File. At this point a new window titled *Untitled* opens into which Matlab commands may be entered. Use the *Save As* option to save the M-file and allocate a name. A suffix .m will be added automatically. The file may be re-opened for editing using the *open* option from the *File* menu.

Exercises

1. Use the *file* menu to create an M-file. Enter a sequence of commands. Save the file using a name of your choice. Execute the newly created M-file. Edit the file if necessary to obtain successful execution, then close the file.
2. Re-open the M-file, make changes to the program and re-execute.

1.6 The *colon* Notation and the *for* Loop

The *colon* operator, which has already been used in specifying rows and columns of matrices (page 4), may be used to specify a range of values. In general the colon operator uses variable names or actual numbers. For example:

1 : n is equivalent to 1, 2, 3, 4, . . . , n. (incrementing by 1)
1 : m : n is equivalent to 1, 1+m, 1+2m, 1+3m, . . . , n. (incrementing by m)

The *colon* notation is particularly useful in the *for* construct, which repeats a sequence of commands a number of times. For example, the following code adds the first 10 integers: 1, 2, . . . , 10, one by one. The running total is held in the variable *addnumbers*.

```
addnumbers = 0;   % initial total
for i = 1 : 10;
  addnumbers = addnumbers + i;   % running total
end
```

Note the use of the % symbol to document the program. Matlab ignores anything on the line following the % sign.

Exercises

1. By making a small change to the program above, use Matlab to find the sum of the even numbers 2, . . . , 10.
2. Create the vector $\mathbf{b} = (2.5 \ \ 5 \ \ 9 \ \ -11)$. Add the elements of **b** by indexing the elements of **b** within a *for* loop, in which a statement of the form

   ```
   eltsum = eltsum + b(i)
   ```

   holds the running total in the variable, *eltsum*.
3. The following series may be used to calculate an approximation to $\sqrt{2}$

   $$x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}, \quad n = 1, 2, \ldots, \quad x_1 = 1.$$

   Calculate $x_1, x_2, \ldots, x_7$ and show convergence to $\sqrt{2}$. The following code would be sufficient

   ```
   x(1) = 1   % first approximation
   for n = 1 : 6
     x(n+1) = x(n)/2 + 1/x(n)   % successive approximations
   end
   ```

   Devise an alternative, more economical version which does not save intermediate results but simply displays and then overwrites the current estimate of $\sqrt{2}$ by the next estimate.
4. The Fibonacci numbers: $0, 1, 1, 2, 3, 5, 8, 13, \ldots$ (each number is the sum of the two preceding numbers) relate to several natural phenomena. Use a *for* loop to print the ratio of successive Fibonacci numbers F_{n+1}/F_n for $n = 2, \ldots, 20$ and so illustrate convergence of this ratio to $(1 + \sqrt{5})/2$ (The Golden ratio).

Note By default Matlab continues printing until a program is completed. To view output page by page enter the command *more on*. To restore the default enter *more*.

1.7 The *if* Construct

In addition to the *loop* structure program flow may be controlled using *if*, *else* and *elseif*. The basic constructs are shown below:

```
if expression;       if expression;       if expression;
  statements;            statements;          statements;
end;                 else                 elseif expression;
                         statements;          statements;
                     end;                 else
                                              statements;
                                          end;
```

where *expression* is a logical statement which evaluates to either *true* or *false* and where *statements* is any set of Matlab statements. Logical statements include variable relationships such as:

== (equals) > (greater than) >= (greater than or equal to)
< (less than) <= (less than or equal to) ~ (not) && (and) || (or)

Use brackets for compound expressions to ensure that Matlab interprets the logic as intended.

Exercises

1. Predict the values of the variables following execution of the fragments of code. Enter the code in a Matlab program to verify the outcome.

(i)
```
x = 2;
y = 1;
if x > 2; y = x; end;
```
(ii)
```
a = 1;
b = 2;
if a == 1; b = a + 1; end;
```
(iii)
```
u = 1; v = 2; w = −2;
if (u ~= 0  &&  u < abs(w) ) || u < 0;
  u = u + w;
else
  u = u − w;
end;
```

2. In using the formula for the solution of a quadratic equation the *discriminant* ($b^2 - 4ac$) may be positive (indicating two real roots), zero (a double root) or negative (complex roots). Write Matlab code to set a variable *roots* to 2, 1 or 0

depending on the value of the *discriminant*. Test the program using various values of a, b and c.

3. Given a vector (or matrix) of dimension (m, n) use a *for loop* within a *for loop* to count the number of non-zero elements. Test your code on simple examples.

Note the function *size* may be used to determine the dimensions of an array. For example [n m] = *size*(A) returns the number of rows, *n* and the number of columns, *m*.

1.8 The *while* Loop

The *while loop* repeats a sequence of instructions until a specified condition is met. It is used as an alternative to the *for loop* in cases where the number of required repeats is not known in advance. The basic construct is:

```
while expression
   Matlab commands
end;
```

If *expression* evaluates to *true* the *Matlab commands* are executed and the program returns to re-evaluating *expression*. If *expression* evaluates to *false* the command following *end;* (if any) is executed.

Exercises

1. Although a *for loop* would be more appropriate, enter the following code which uses a *while* loop to find the sum of the numbers 1, ..., 10.

```
addnumbers = 0;  % set running total to 0
n = 1;  % first number
while n <= 10
  addnumbers = addnumbers + n;  % add current number to running total
  n = n + 1;  % next number
end;
addnumbers  % show the result
```

2. Create a vector of numbers which includes one or more zeros. Construct a *while loop* to find the sum of the numbers up to the first zero.
3. Use a *while loop* and the previously quoted series to calculate an approximation to $\sqrt{2}$ accurate to 4 decimal places by terminating the loop when the difference between successive estimates is less than 0.00005.

Note Enter *Ctrl C* from the keyboard to terminate execution prematurely (useful if a program runs out of control).

1.9 Simple Screen Output

As an alternative to entering the name of a variable the function *disp* may be used to output a value or a string. The standard forms are *disp*(x) and *disp*('*s*') where x is a variable and s is a string of alphanumeric and other characters.

Exercises

1. Assign a value to the single variable *root* and enter the command

```
disp('Root of the equation:');    disp(root);
```

   to show how more meaningful output may be produced.
2. Repeat the previous question with a vector rather than a single variable.

1.10 Keyboard Input

The function *input* may be used to request user input. The standard form is x = *input*('*request*'). The user sees the character string *request* on the screen and enters number(s) from the keyboard to be allocated to the program variable x. If an answer in the form of a character string is required the command has the form x = *input*('*request*','*s*'). The command is useful for constructing programs that interact with the user.

Note The character string, \n may be inserted into the *request* string to move the cursor to a new line.

Exercises

1. As part of an interactive program, use the following code to obtain the value of m and echo the input to the screen. The program asks for confirmation (Y or N) that the number is correct and if necessary repeats the original request.

```
m=input('Enter the number of data values \n');
reply=input('Is that correct, answer Y or N\n','s');
while ~ strcmp(reply,'Y')
  input('Enter the number of data values\n');
  reply=input('Is that correct, answer Y or N\n','s');
end;
disp('Thank you');
```

2. Modify the program by allowing a maximum of 3 corrections to the first entry.
3. Write a program to request a password from the user. Compare the reply with a stored string. If necessary repeat the request at most twice until there is agreement.

Note the function *strcmp* may be used to compare two strings. *strcmp*(a, b) evaluates to *true* if *a* and *b* are identical and *false* otherwise. If two strings have equal leading characters but one has trailing blanks they are not considered equal. The function *strncmp* for comparing the leading *n* characters of two strings is also provided by Matlab. Enter the command *help strncmp* for details.

1.11 User Defined Functions

Matlab provides a variety of built-in functions to cover a vast range of activities but it also allows user-defined functions. As we will see this can be a very useful facility. Each user-defined file must be stored in an M-file with the same name (apart from the .m suffix) as the function. The function heading has the form

$$\textit{function}\ [\textit{out}1, \textit{out}2, \ldots] = \textit{functionname}\ (\textit{in}1, \textit{in}2, \ldots)$$

where *functionname* is the chosen name of the function (and the M-file), $in1, in2, \ldots$ are input parameters to the function and $out1, out2, \ldots$ are output parameters. In practice actual values or names of variables would be used as parameters.

As an example we consider a function for calculating the sum of the series

$$1 + x + x^2 + \cdots + x^n.$$

We will call our function *sumGP* and allow for two input variables *x* and *n* and one output parameter *sum*.

The following code (or similar) would be entered in an M-file with the name *sumGP.m*. The variable names we have used as parameters (and corresponding variables within the function) are an arbitrary choice.

```
function [result] = sumGP( x, n )
  if x == 1;  result = n+1    % Check for the exceptional case.
  else result = (x∧(n+1) − 1)/(x − 1)
  end
```

Assuming the M-file, *sumGP.m* is in the working directory a command of the form $z = sumGP(p, q)$ or $sumGP(p, q)$ would be used to apply the function where *p* and *q* are either variable names or actual values. In this instance variables *z*, *p* and *q* would take the place of parameters *result*, *x* and *n* used in the formal definition of *SumGP*.

Exercises

1. Create the aforementioned function *sumGP* and test it using the commands *sumGP*(1, 4) (which should return the value 5) and *sumGP*(0.5, 2) (to return 1.75).
2. Write and test a function FtoC having one input and one output parameter to convert degrees Fahrenheit to degrees Centigrade. Use the formula $C = (5/9)(F - 32)$. Establish the function in the M-file *FtoC.m* and test it from a program with calls such as *FtoC*(32) and *centigrade* = *FtoC*(212).

3. Write a function *MaxElement* to find the largest absolute value of all elements of an array or vector. Test the function on simple examples. Note that for a given matrix or vector, A the Matlab function *max* applied to $abs(A)$, as in $max(abs(A))$ would produce the same result.

1.12 Basic Statistics

A list of basic statistical functions, including *mean* and *std* (standard deviation), and plotting functions *bar* (bar charts) and *pie* (pie charts) may be found by following the links *Help* → *Product Help* → *Functions: By Category* → *Data Analysis* from the command window. Other statistical applications including Regression Analysis, Hypothesis Testing and Multivariate Methods may be found in the Statistics Toolbox. Follow the links *Help* → *Product Help* → *Statistics Toolbox* for details.

Exercises

1. As examples of some of the basic functions store a random selection of numbers in a vector v and enter each of the commands: $mean(v)$, $std(v)$, $bar(v)$ and $pie(v)$ separately.
2. Enter the command *help pie* for an explanation of the *pie* function and use the information to add labels to the pie chart of variables v.

1.13 Plotting

For 2-D graphics, the basic commands are: $plot(v)$, $plot(u, v)$ and $plot(u, v, 's')$ where u and v are vectors and s is a combination of up to three character options, one from under each heading of Table 1.1.

The default option for plotting corresponds to '−b' (solid line, no data marks, blue).

Exercises

1. Store a random selection of (x, y) coordinates in the vectors x and y and enter the command $plot(x, y, ' - +r')$.
2. Repeat the previous question with other character strings.

Table 1.1 Plotting options

Line style		Data markers		Colour			
−	solid line	+	plus sign	y	yellow	r	red
−.	dash–dot line	o	circle	m	magenta	g	green
−−	dashed line	.	dot	c	cyan	b	blue
:	dotted line	⋆	asterisk	k	black	g	white
		x	cross				

3. Use the following program to plot the functions $y = x^2$ over the interval $0 <= x <= 4$.

```
% define sufficient x-points across the range [0, 4] to produce a smooth curve
% use Matlab function linspace to generate 40 equally spaced values across
% the range [0, 4]
   x = linspace ( 0, 4, 40 );
% form the vector of y-values, using the .* operator for element by element
% matrix multiplication
   y = x.*x
% plot the curve using continuous (red) lines,
   plot(x, y, 'r');
```

4. Plot the functions $y = x$ and $y = sin(x)$ and $y = cos(x)$ over the range $[0, 2\pi]$ on the same graph but using different colours. The command *hold* may be used to retain the current plot to add further plots. Re-issuing the command *hold* releases the current plot.
5. Details of the enormous range of Matlab facilities for plotting, annotation and visualisation may be found by following the links from the help menu in the command window. As a rather spectacular example use the following program to plot the three dimensional surface defined by $z = x^2 - y^2$ across the x–y ranges $[-2, 2]$ on both axes.

```
% establish a grid of points equally spaced at intervals of 0.1 across the range
   [x, y] = meshgrid(−2 : 0.1 : 2,  −2 : 0.1 : 2);
% establish z-values using element by element multiplication
   z = x.*x − y.*y
% plot the surface
   surf(x, y, z);
   colorbar  % show the colour scale.
```

1.14 Formatted Screen Output

By default Matlab prints numbers using 5 digits. To obtain output using up to 16 digits enter the command *format long*. To restore the default enter *format short*. In this section we consider how numbers may be presented in more specialised form in terms of style, field width and number of decimal places. In general formatted output of one or more variables is produced by using the command *sprintf* to build an output string, which is displayed using *disp*.

The general form is
sprintf('*formatstring*', *variables*)
where *formatstring* is a string composed of *text*, *formats* and *parameters*
variables is a list of variables to be output and separated by commas.
If necessary the *formatstring* is repeated until the list of the variables to be output is completed.

Formats are preceded by % and are applied in the order of the variables. The more used formats are illustrated below:

d	for integer format for example %6d for a field width of 6 places.
f	fixed point format for example %10.5f for a field width of 10 places and 5 decimal places.
e	floating point (exponent) format for example %10.3e for a field width of 10 places and 3 decimal place exponent.

We will use the parameter \\n (newline) but there are others. For more details and details of other features of the format command, enter *help format* in the command window.

Exercises

1. Enter the following code to compare π and its popular approximation 22/7.

```
ratio = 22/7;
s1 = sprintf('pi to 6 decimal places is %8.6f', pi);
s2 = sprintf('22/7 to 6 decimal places is %8.6f', ratio);
disp(s1); disp(s2);
```

2. Repeat the previous question making use of the \n parameter to obtain the same output using a single string. It may be necessary to use more that one line to enter the format. Use the continuation mark as shown in the example below.
3. Allocate values to variables *Node*, *ux* and *uy* and use the following commands to produce the output:
   Node 27 ux 1.0934 uy 3.5e+005

```
s = sprintf( ...   % three dots indicate a continuation to the next line
'Node %3d ux %6.4f uy %10.3e', Node, ux, uy);
disp(s);
```

4. Display the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 4.56 \\ 3 & 4 & 5.0 \\ 3 & 29 & 4.567 \end{pmatrix}$$

   noting that Matlab stores elements of a matrix in column order and so to obtain the desired effect the transpose of **A** is used, for example:

```
s = sprintf('%6.3f  %6.3f  %6.3f  \n ', A');
disp(s);
```

5. Repeat the previous question but with a more detailed format so that the display of matrix A bears a closer resemblance to the form shown above.

1.15 File Input and Output

In order to access a file it must be opened. A command of the form *fid* = *fopen*('*filename*', '*w*+') opens a file for read, write and create access in M/S Windows systems (other operating systems may vary). Formats may be used for appending output to an existing file. Enter the command *help fopen* for details. The variable, *fid* is known as a file identifier and is used by Matlab to identify the file in subsequent operations.

1.15.1 Formatted Output to a File

The command *fprintf* writes formatted data to a file. The structure of the command is *exactly* the same as that of *sprintf* but with a file identifier as the first parameter. For example *fprintf*(*fid*, . . .).

Exercises

1. Repeat any of the questions from (1.14) but print the results to a file. Display the file *after* executing the formatting code using the command *type filename*.
   Use the following code as an example which opens and writes to a Windows file *myfile.txt*. If the file already exists it is overwritten.

```
fid = fopen('myfile.txt','w+');
fprintf(fid, 'pi to 6 decimal places is %8.6f ', pi);
type myfile.txt
```

1.15.2 Formatted Input from a File

There are other commands for reading formatted input from a file but *textscan* is particularly useful as it does not require the precise specification of incoming formats. For example to read a line of the form

```
Node 27 ux 1.0934 uy 3.5e+005
```

textscan only requires the information

```
%s %d %s %f %s %f
```

where %s denotes a character string, %d denotes an integer and %f denotes a fixed or floating point number. Other formats are available.

Exercises

1. Create a file *data.txt* with first line:

```
Node 27 ux 1.0934 uy 3.5e+005
```

Enter the code shown below to open the file, read the data and allocate the numbers to program variables a, b and c. The function *textscan* produces output in a Matlab *cell*, which differs from an array in that elements of different sizes and different types may be stored. In this example the name *mydata* has been chosen for the *cell*, output corresponding to each item in the format string is placed in successive positions of *mydata*. Whereas items from a matrix are retrieved using enclosing (and) parentheses, items from a cell are retrieved using the pair { and }.

```
fid = fopen('data.txt');
mydata = textscan(fid, ' %s %d %s %f %s %f ')
% retrieve the numbers 27, 1.0934 and 3.5e+005
% the strings Node, ux and uy will be held in mydata{1}, mydata{3} and
% mydata{5}
a = mydata{2}
b = mydata{4}
c = mydata{6}
```

The function *textscan* may be used to collect data from several consecutive lines of data provided the stated format applies. To limit the number of lines that textscan reads for a given format a third parameter is available. A command of the form *textscan(fid, format, n)* would apply the format n times from the current position. A command *frewind(fid)* is available to re-set the internal pointer to start of the file.

A further feature of *textscan* allows unwanted data to be skipped by inserting an $*$ between the relevant % and the following s, d or f in the format string.

Exercises

1. Add a few more lines of similar data to the file *mydata*. Use a single *textscan* command to read the whole file and notice how the data is retrieved as column vectors.
2. Modify the format string of the previous question. Choose a selection of columns of the data file to be retrieved.

1.15.3 Unformatted Input and Output (Saving and Retrieving Data)

Workspace variables may be stored more economically as unformatted data (data in binary form and as such impossible to read with the human eye) in a file having a name with a *.mat* suffix. Use the commands *save* to save and *load* to restore.

For example the command *save('savedata.mat'*, 'x', 'y', 'z') would store variables x, y, z in the file *savedata.mat* and *load('savedata.mat'*, 'x', 'y', 'z') would re-load the variables. The commands *save* and *load* without parameters save and load all workspace variables.

Exercises

1. Assign variables to matrices A and B. Save the matrices in unformatted form to a *.mat* file.
2. Clear A and B from the workspace using the command *clear A B*. Check that they no longer exist.
3. Retrieve the matrices A and B from the *.mat* file. Check that nothing has been lost.

Chapter 2
Linear Equations

Aims In this chapter we look at ways of solving systems of linear equations, sometimes known as *simultaneous* linear equations. We investigate

- Gaussian elimination and its variants, to obtain a solution to a system.
- partial pivoting *within* Gaussian elimination, to improve the accuracy of the results.
- iterative refinement *after* Gaussian elimination, to improve the accuracy of a first solution.

In some circumstances the use of Gaussian elimination is not recommended and other techniques based on iteration are more appropriate including

- Gauss–Seidel iteration.

We look at the method in detail and compare this iterative technique with direct solvers based on elimination.

Overview We begin by examining what is meant by a linear relationship, before considering how to handle the (linear) equations which represent such relationships. We describe methods for finding a solution, which simultaneously satisfies several such linear equations. In so doing we identify situations for which we can guarantee the existence of one, and only one, solution; situations for which more than one solution is possible; and situations for which there may be no solution at all.

As already indicated, the methods to be considered neatly fall into two categories; direct methods based on elimination; and iterative techniques. We look at each type of method in detail. For direct methods we need to bear in mind the limitations of computer arithmetic, and hence we consider how best to implement the methods so that as the algorithm proceeds the accumulation of error may be kept to a minimum. For iterative methods the concern is more related to ensuring that convergence to a specified accuracy is achieved in as few iterations as possible.

Acquired Skills After reading this chapter you will appreciate that solving linear equations is not as straightforward a matter as might appear. No matter which method you use accumulated computer errors may make the true solution unattainable. However you will be equipped with techniques to identify the presence of significant errors and to take avoiding action. You will be aware of the differences between a variety of direct and indirect methods and the circumstances in which the use of a particular method is appropriate.

2.1 Introduction

One of the simplest ways in which a physical entity may be seen to behave is to vary in a linear manner with respect to some external influence. For example, the length (L) of the mercury column in a thermometer is accepted as being directly related to ambient temperature (T); an increase in temperature is shown by a corresponding increase in the length of the column. This linear relationship may be expressed as

$$L = kT + c.$$

Here k and c are constants whose values depend on the units in which temperature (degrees Celsius, Fahrenheit, etc.) and length (millimetres, inches, etc.) are measured, and on the bore of the tube. k represents the change in length relative to a rise in temperature, whilst c corresponds to the length of the column at zero degrees. We refer to T as the **independent variable** and L as the **dependent variable**. If we were to plot values of L against corresponding values of T we would have a straight line with slope k and intercept on the L axis equal to c.

A further example of a linear relationship is Hooke's Law (2.1) which relates the tension in a spring to the length by which it has been extended. If T is the tension, x is the extension of the spring, a is the unextended length and λ is the modulus of elasticity, then we have

$$T = \frac{\lambda}{a}x. \tag{2.1}$$

Typical units are Newtons (for T and λ) and metres (for x and a). A plot of T against x would reveal a straight line passing through the origin with slope λ/a.

An example which involves more than two variables is supplied by Kirchoff's Law which states that the algebraic sum of currents meeting at a point must be zero. Hence, if i_1, i_2 and i_3 represent three such currents, we must have

$$i_1 + i_2 + i_3 = 0. \tag{2.2}$$

An increase in the value of one of the variables (say i_1) must result in a corresponding decrease (to the same combined value) in one or more of the other two variables (i_2 and/or i_3) and in this sense the relationship is linear. For a relationship to be linear, the variables (or constant multiples of the variables) are combined by additions and subtractions.

2.2 Linear Systems

In more complicated modelling situations it may be that there are many linear relationships, each involving a large number of variables. For example, in the stress analysis of a structure such as a bridge or an aeroplane, many hundreds (or indeed thousands) of linear relationships may be involved. Typically, we are interested in determining values for the dependent variables using these relationships so that we can answer the following questions:

- For a given extension of a spring, what is the tension?
- If one current has a specific value, what values for the remaining currents ensure that Kirchoff's Law is preserved?
- Is a bridge able to withstand a given level of traffic?

As an example of a linear system consider a sporting event where spectators were charged for admission at the rate of £15.50 for adults and £5.00 for concessions. It is known that the total takings for the event was £3312 and that a total of 234 spectators were admitted to the event. The problem is to determine how many adults watched the event.

If we let x represent the number of adults and y the number of concessions then we know that the sum of x and y must equal 234, the total number of spectators. Further, the sum of $15.5x$ and $5y$ must equal 3312, the total takings. Expressed in mathematical notation, we have

$$x + y = 234 \tag{2.3}$$

$$15.5x + 5y = 3312. \tag{2.4}$$

Sets of equations such as (2.3) and (2.4) are known as systems of linear equations or **linear systems**. It is an important feature of this system, and other systems we consider in this chapter, that the number of equations is equal to the number of unknowns to be found.

When the number of equations (and consequently the number of variables in those equations) becomes large it can be tedious to write the system out in full. In any case, we need some notation which will allow us to specify an arbitrary linear system in compact form so that we may conveniently examine ways of solving such a system. Hence we write linear systems using matrix and vector notation as

$$\mathbf{Ax} = \mathbf{b} \tag{2.5}$$

where $\mathbf{A}$ is a matrix, known as the **coefficient matrix** (or **matrix of coefficients**), $\mathbf{b}$ is the vector of right-hand sides and $\mathbf{x}$ is the vector of unknowns to be determined. Assuming n (the length of $\mathbf{x}$) unknowns to be determined, $\mathbf{b}$ must be an n-column vector and $\mathbf{A}$ an $m \times n$ square matrix, although for the time being we only consider systems with the same number of equations as unknowns, $(m = n)$.

Problem

Write (2.3), (2.4) in matrix–vector form.

Solution

We have

$$\mathbf{Ax} = \mathbf{b}, \quad \text{where } \mathbf{A} = \begin{pmatrix} 1 & 1 \\ 15.5 & 5 \end{pmatrix}, \ \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}, \ \mathbf{b} = \begin{bmatrix} 234 \\ 3312 \end{bmatrix}.$$

A great number of electronic, mechanical, economic, manufacturing and natural processes may be modelled using linear systems in which, as in the above simple example, the number of unknowns matches the number of equations. If, as sometimes happens, the modelling process results in nonlinear equations (equations involving perhaps squares, exponentials or products of the unknowns) it is often the case that the iterative techniques involved to solve these systems reduce to finding the solution of a linear system at each iteration. Hence linear equation solvers have a wider applicability than might at first appear to be the case.

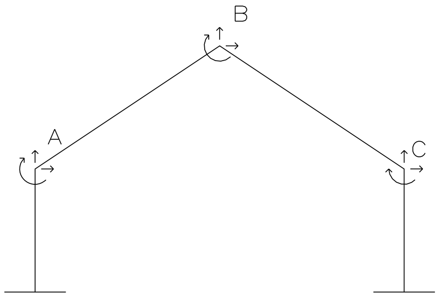
Since the modelling process either directly or indirectly yields a system of linear equations, the ability to solve such systems is fundamental. Our interest is in methods which can be implemented in a computer program and so we must bear in mind the limitations of computer arithmetic.

Problem

The structure shown in Fig. 2.1 shows a pitched portal frame. Using standard engineering analysis and assuming certain restrictions on the dimensions of the structure (the details of which need not concern us) it can be shown that the resulting displacements and rotations at the points A, B and C may be related to the acting forces and moments by a set of nine linear equations in nine unknowns. The equations involve constants reflecting the nature of the material of the structure.

Writing the displacements and rotations in the ascending sequence $x_1, x_2, \ldots, x_9$ and assigning values to the constants and the external loads (the forces and the

Fig. 2.1 Pitched portal frame



moments) we might find that in a particular instance the equations take the following form:

$$\begin{aligned}
x_1 &&&= 0 \quad (2.6)\\
x_2 &&&= 10 \quad (2.7)\\
2x_3 - 10x_5 + 3x_6 &&&= 5 \quad (2.8)\\
3x_4 - 2x_5 + 2x_6 - 3x_7 &&&= 2 \quad (2.9)\\
x_5 + 5x_6 + x_7 &&&= 14 \quad (2.10)\\
3x_6 - 4x_7 &&&= 17 \quad (2.11)\\
2x_7 &&&= -4 \quad (2.12)\\
2x_8 &&&= 10 \quad (2.13)\\
x_9 &&&= 1. \quad (2.14)
\end{aligned}$$

Solution

Although this is quite a large system we can take advantage of its special structure to obtain a solution fairly easily. From (2.14) that $x_9 = 1$, from (2.13) that $x_8 = 5$, and from (2.12) that $x_7 = -2$.

To obtain the rest of the solution values we make use of the values that we already know. Substituting the value of x_7 into (2.11) we have $3x_6 + 8 = 17$ and so $x_6 = 3$. Now that we know x_6 and x_7 we can reduce (2.10) to an equation involving x_5 only, to give $x_5 + 15 - 2 = 14$, and so $x_5 = 1$. Similarly we can determine x_4 from (2.9) using $3x_4 - 2 + 6 + 6 = 2$, to give $x_4 = -2\frac{2}{3}$. Finally, to complete the solution we substitute the known values for x_5 and x_6 into (2.8) to obtain $2x_3 - 10 + 9 = 5$ and conclude that $x_3 = 3$. From (2.7) and (2.6) we have $x_2 = 10$ and $x_1 = 0$ to complete the solution.

To summarise, we have $x_1 = 0$, $x_2 = 10$, $x_3 = 3$, $x_4 = -2\frac{2}{3}$, $x_5 = 1$, $x_6 = 3$, $x_7 = -2$, $x_8 = 5$ and $x_9 = 1$.

Discussion

The system of equations we have just solved is an example of an **upper triangular system**. If we were to write the system in the matrix and vector form $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}$ would be an **upper triangular matrix**, that is a matrix with zero entries below the diagonal. Mathematically the matrix $\mathbf{A}$ is upper triangular if for all elements a_{ij}

$$a_{ij} = 0, \quad i > j.$$

For systems in which the coefficient matrix is upper triangular the last equation in the system is easily solved and this value can then be used to solve the penultimate equation. The results from these last two equations may be used to solve the previous equation, and so on. In this manner the whole system may be solved. The process of working backwards to complete the solution is known as **backward substitution**.

2.3 Gaussian Elimination

As part of the process we consider how general linear systems may be solved by reduction to upper triangular form.

Problem

Find the numbers of adults and concessions at the sporting event (2.3), (2.4) by solving the system of two equations in two unknowns.

Solution

In subtracting 15.5 times (2.3) from (2.4) we have $1.5y = 91.5$, and combining this with the original first equation we have

$$x + y = 234 \tag{2.15}$$

$$-10.5y = -315 \tag{2.16}$$

which is in upper triangular form. Solving (2.16) gives $y = 30$ (concessions) and substituting this value into (2.15) gives $x = 234 - 30 = 204$ (adults).

Discussion

The example illustrates an important mechanism for determining the solution to a system of equations in two unknowns, which may readily be extended to larger systems. It has two components:

1. Transform the system of equations into an equivalent one (that is, the solution to the transformed system is mathematically identical to the solution to the original system) in which the coefficient matrix is upper triangular.
2. Solve the transformed problem using backward substitution.

A linear system may be reduced to upper triangular form by means of successively subtracting multiples of the first equation from the second, third, fourth and so on, then repeating the process using multiples of the newly-formed second equation, and again with the newly-formed third equation, and so on until the penultimate equation has been used in this way. Having reduced the system to upper triangular form backward substitution is used to find the solution. This systematic approach, known as **Gaussian elimination**, is illustrated further in the following example.

Problem

Solve the following system of four equations in four unknowns

$$2x_1 + 3x_2 + 4x_3 - 2x_4 = 1 \tag{2.17}$$

$$x_1 - 2x_2 + 4x_3 - 3x_4 = 2 \tag{2.18}$$

$$4x_1 + 3x_2 - x_3 + x_4 = 2 \tag{2.19}$$

$$3x_1 - 4x_2 + 2x_3 - 2x_4 = 5. \tag{2.20}$$

Solution

As a first step towards an upper triangular system we eliminate x_1 from (2.18), (2.19) and (2.20). To do this we subtract $1/2$ times (2.17) from (2.18), 2 times (2.17) from (2.19), and $3/2$ times (2.17) from (2.20). In each case the fraction involved in the multiplication (the **multiplier**) is the ratio of two coefficients of the variable being eliminated from (2.18)–(2.20). The numerator is the coefficient of the variable to be eliminated and the denominator is the coefficient of that variable in the equation which is to remain unchanged. As a result of these computations the original system is transformed into

$$2x_1 + 3x_2 + 4x_3 - 2x_4 = 1 \tag{2.21}$$

$$-\tfrac{7}{2}x_2 + 2x_3 - 2x_4 = \tfrac{3}{2} \tag{2.22}$$

$$-3x_2 - 9x_3 + 5x_4 = 0 \tag{2.23}$$

$$-\tfrac{17}{2}x_2 - 4x_3 + x_4 = \tfrac{7}{2}. \tag{2.24}$$

To proceed we ignore (2.21) and repeat the process on (2.22)–(2.24). We eliminate x_2 from (2.23) and (2.24) by subtracting suitable multiples of (2.22). We take $(-3)/(-\frac{7}{2})$ times (2.22) from (2.23) and $(-17)/(-7)$ times (2.22) from (2.24) to give (after the removal of common denominators) the system

$$2x_1 + 3x_2 + 4x_3 - 2x_4 = 1 \tag{2.25}$$

$$-\tfrac{7}{2}x_2 + 2x_3 - 2x_4 = \tfrac{3}{2} \tag{2.26}$$

$$-\tfrac{75}{7}x_3 + \tfrac{47}{7}x_4 = -\tfrac{9}{7} \tag{2.27}$$

$$-\tfrac{62}{7}x_3 + \tfrac{41}{7}x_4 = -\tfrac{1}{7}. \tag{2.28}$$

Finally we subtract $62/75$ times (2.27) from (2.28) to give the system

$$2x_1 + 3x_2 + 4x_3 - 2x_4 = 1 \tag{2.29}$$

$$-\tfrac{7}{2}x_2 + 4x_3 - 2x_4 = \tfrac{3}{2} \tag{2.30}$$

$$-\tfrac{75}{7}x_3 + \tfrac{47}{7}x_4 = -\tfrac{9}{7} \tag{2.31}$$

$$\tfrac{161}{525}x_4 = -\tfrac{483}{525}. \tag{2.32}$$

We now have an upper triangular system which may be solved by backward substitution. From (2.32) we have $x_4 = 3$. Substituting this value in (2.31) gives $x_3 = 2$. Substituting the known values for x_4 and x_3 in (2.30) gives $x_2 = -1$. Finally using (2.29) we find $x_1 = 1$. The extension of Gaussian elimination to larger systems is straightforward. The aim, as before, is to transform the original system to upper triangular form which is solved by backward substitution.

Discussion

The process we have described may be summarised in matrix form as pre-multiplying the coefficient matrix **A** by a series of lower-triangular matrices to form an upper-triangular matrix.

We have

$$\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&-\frac{62}{75}&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&-3/(\frac{7}{2})&1&0\\0&-\frac{17}{7}&0&1\end{pmatrix}\begin{pmatrix}1&0&0&0\\-\frac{1}{2}&1&0&0\\-2&0&1&0\\-\frac{3}{2}&0&0&1\end{pmatrix}\begin{pmatrix}2&3&4&-2\\1&-2&4&-3\\4&3&-1&1\\3&-4&2&-2\end{pmatrix}$$

$$=\begin{pmatrix}2&3&4&-2\\&-\frac{7}{2}&4&-2\\&&-\frac{75}{7}&\frac{47}{7}\\&&&\frac{161}{525}\end{pmatrix}$$

where it can be seen that sub-diagonal entries correspond to the multipliers in the elimination process. Since the product of lower triangular matrices is also lower triangular we may write the above as $\mathbf{LA} = \mathbf{U}$ where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is upper triangular. Since the inverse of a lower triangular matrix is also lower triangular multiplying both sides by the inverse of $\mathbf{L}$ produces

$$\mathbf{A} = \mathbf{LU}$$

where $\mathbf{L}$ is another lower triangular matrix and $\mathbf{U}$ is upper triangular. This factorisation is generally known as the **LU decomposition**. If the factorisation is known it may be used repeatedly to solve the same set of equations for different right hand sides, as will be shown in Sect. 2.3.3.

2.3.1 Row Interchanges

In certain circumstances, Gaussian elimination fails as we see in the following example.

Problem

Solve the system of 4 equations in 4 unknowns

$$2x_1 - 6x_2 + 4x_3 - 2x_4 = 8 \tag{2.33}$$
$$x_1 - 3x_2 + 4x_3 + 3x_4 = 6 \tag{2.34}$$
$$4x_1 + 3x_2 - 2x_3 + 3x_4 = 3 \tag{2.35}$$
$$x_1 - 4x_2 + 3x_3 + 3x_4 = 9. \tag{2.36}$$

Solution

To eliminate x_1 from (2.34), (2.35) and (2.36) we take multiples $\frac{1}{2}$, 2 and $\frac{1}{2}$ of (2.33) and subtract from (2.34), (2.35) and (2.36) respectively. We now have the linear system

$$\begin{aligned}
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\
2x_3 + 4x_4 &= 2 && (2.37)\\
15x_2 - 10x_3 + 7x_4 &= -13 && (2.38)\\
-x_2 + x_3 + 4x_4 &= 5. && (2.39)
\end{aligned}$$

Since x_2 does not appear in (2.37) we are unable to proceed as before. However a simple re-ordering of the equations enables x_2 to appear in the second equation of the system and this permits further progress. Exchanging (2.37) and (2.38) we have the system

$$\begin{aligned}
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\
15x_2 - 10x_3 + 7x_4 &= -13 && (2.40)\\
2x_3 + 4x_4 &= 2 \\
-x_2 + x_3 + 4x_4 &= 5. && (2.41)
\end{aligned}$$

Resuming the elimination procedure we eliminate x_2 from (2.41) by taking a multiple $(-1)/15$ of (2.40) from (2.41) to give

$$\begin{aligned}
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\
15x_2 - 10x_3 + 7x_4 &= -13 \\
2x_3 + 4x_4 &= 2 && (2.42)\\
\tfrac{1}{3}x_3 + \tfrac{67}{15}x_4 &= \tfrac{62}{15}. && (2.43)
\end{aligned}$$

Finally, to eliminate x_3 from (2.43) we take a multiple $1/6$ of (2.42) from (2.43). As a result we now have the linear system

$$\begin{aligned}
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\
15x_2 - 10x_3 + 7x_4 &= -13 \\
2x_3 + 4x_4 &= 2 \\
\tfrac{57}{15}x_4 &= \tfrac{57}{15},
\end{aligned}$$

which is an upper triangular system that can be solved by backward substitution to give the solution $x_4 = 1$, $x_3 = -1$, $x_2 = -2$, and $x_1 = 1$.

Table 2.1 Gaussian elimination with row interchanges

	Solve the $n \times n$ linear system $\mathbf{Ax} = \mathbf{b}$ by Gaussian elimination
1	Reduce to upper triangular form: For $i = 1, 2, \ldots, n-1$
1.1	Ensure $a_{ii} \neq 0$. Exchange rows (including b_i) if necessary
1.2	Subtract multiples of row i (including b_i) from rows $i+1, i+2, \ldots, n$ so that the coefficient of x_i is eliminated from those rows
2	Backward substitution: a_{ij} and b_i are current values of the elements of $\mathbf{A}$ and $\mathbf{b}$ as the elimination proceeds
2.1	$x_n = b_n / a_{nn}$
2.2	For $i = n-1, n-2, \ldots, 1$, $x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} x_j) / a_{ii}$

Discussion

The system (2.33)–(2.34) introduced the complication of a zero appearing as a leading coefficient (the **pivot**) in the equation (the **pivotal equation**) which would normally be used to eliminate the corresponding unknown from all subsequent equations. This problem was easily overcome by a re-arrangement which ensured a non-zero pivot. The method of Gaussian elimination with row interchanges is summarised in Table 2.1.

2.3.2 Partial Pivoting

In the previous section we saw that a re-arrangement of the equations is sometimes necessary if Gaussian elimination is to yield an upper triangular system. Here we show that re-arrangements of this form can be useful in other ways.

Up to this point we have used exact arithmetic to solve the equations, even though at times the fractions involved have been a little cumbersome. In real life of course we would use a computer to perform the arithmetic. However, a computer does not normally carry out arithmetic with complete precision. The difference between accurate and computer arithmetic is known as **rounding error**.

Unless specified otherwise Matlab uses arithmetic conforming to IEEE[1] standards by which numbers are represented by the formula $(-1)^{\mathrm{s}} 2^{(\mathrm{e}-1023)}(1+\mathrm{f})$ using a 64-bit word. This is traditionally known as double precision arithmetic. The sign, s of the number is held in a 1-bit word, an eleven-bit binary number holds the exponent, e and a fifty-two bit binary number called the mantissa holds the precision, f of the number which can range from 1 to 2046.

We consider two problems, each of which is a system of two equations in two unknowns. They are, in fact, the same problem mathematically; all that we have

[1] Institute of Electrical and Electronics Engineers

done is to reverse the order of the two equations. The solution can be seen to be $x_1 = 1$, $x_2 = 1$. To illustrate the effect of rounding error we consider the following problem.

Problem

Solve the following systems rounding the results of all arithmetic operations to three significant figures.

System (1):

$$0.124x_1 + 0.537x_2 = 0.661 \tag{2.44}$$

$$0.234x_1 + 0.996x_2 = 1.23 \tag{2.45}$$

System (2):

$$1.234x_1 + 0.996x_2 = 1.23 \tag{2.46}$$

$$0.124x_1 + 0.537x_2 = 0.661. \tag{2.47}$$

Solution

- System (1): In the usual manner we eliminate x_2 from (2.45) by subtracting the appropriate multiple of (2.44). In this case the multiple is 0.234/0.124 which, rounded to three significant figures, is 1.89. Multiplying 0.537 by 1.89 gives 1.01 and subtracting this from 0.996 gives −0.0140. Similarly for the right-hand side of the system, 0.661 multiplied by 1.89 is 1.25 and when subtracted from 1.23 gives −0.0200. The system in its upper triangular form is

$$0.124x_1 + 0.537x_2 = \quad 0.661 \tag{2.48}$$

$$-\,0.014x_2 = -0.020. \tag{2.49}$$

  In backward substitution, x_2 is obtained from $\frac{0.020}{0.014}$ to give 1.43 which is used in (2.48) to give $x_1 = -0.863$. In this case the solution obtained by using arithmetic which retains three significant figures is nothing like the true solution.
- System (2): We eliminate x_2 from (2.47) by subtracting the appropriate multiple of (2.46). In this case the multiple is 0.124/0.234, which is 0.530 to three significant figures. Multiplying 0.537 by 0.530 and subtracting from 0.996 gives 0.00900. Multiplying 0.661 by 0.530 and subtracting from 1.23 also gives 0.00900 and so the system in upper triangular form is

$$0.234x_1 + 0.996x_2 = 1.23$$

$$0.009x_2 = 0.009.$$

  By backward substitution we have x_2 = 1.00, followed by x_1 = 1.00 which is the exact solution.

Discussion

We can infer from these two simple examples that the order in which equations are presented can have a significant bearing on the accuracy of the computed solution. The example is a little artificial, given that a modern computer carries out its arithmetic to many more significant figures than the three we have used here, often as many as sixteen or more. Nevertheless, for larger systems it may be verified that the order in which the equations are presented has a bearing on the accuracy of a computer solution obtained using Gaussian elimination. The next example describes a strategy for rearranging the equations within Gaussian elimination with a view to reducing the effect of rounding errors on a computed solution. We note in passing that an **LU** decomposition which takes account of partial pivoting of a non-singular matrix **A** is available in the form $\mathbf{PA} = \mathbf{LU}$ where **L** and **U** are lower and upper triangular matrices and **P** is a permutation matrix, a matrix of zeros and ones that in pre-multiplying **A** performs the necessary row exchanges. For example exchanging rows 2 and 3 of a 3×3 matrix **A** may be achieved by pre-multiplying **A** by the permutation matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

Problem

Solve the system (2.33)–(2.36) using row exchanges to minimise pivotal values at each step of the elimination.

Solution

We recall that in obtaining the previous solution example it was necessary to rearrange the equations during the reduction to upper triangular form to avoid a zero pivot. In this example we perform similar rearrangements, not just out of necessity, but also with accuracy in mind. Looking down the first column we see that the largest (in magnitude) coefficient of x_1 occurs in (2.35). We thus re-order the equations to make this the pivot. This involves exchanging (2.35) and (2.33). We now have the linear system

$$4x_1 + 3x_2 - 2x_3 + 3x_4 = 3 \tag{2.50}$$

$$x_1 - 3x_2 + 4x_3 + 3x_4 = 6 \tag{2.51}$$

$$2x_1 - 6x_2 + 4x_3 - 2x_4 = 8 \tag{2.52}$$

$$x_1 - 4x_2 + 3x_3 + 3x_4 = 9 \tag{2.53}$$

and we eliminate x_1 from (2.51), (2.52) and (2.53) by subtracting multiples $\frac{1}{4}$, $\frac{2}{4}$ and $\frac{1}{4}$ of (2.50) from (2.51), (2.52) and (2.53) respectively. Looking back to the system (2.33)–(2.36) we see that the multipliers without re-arrangement were $\frac{1}{2}$, 2 and $\frac{1}{2}$. Now they are uniformly smaller. This will prove to be significant. At the end of this, the first stage, of elimination, the original system is transformed to

$$\begin{aligned}
4x_1 - \ 3x_2 - \ 2x_3 + \ 3x_4 &= 3 \\
- 3\tfrac{3}{4}x_2 + 4\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 5\tfrac{1}{4} \qquad (2.54)\\
- 7\tfrac{1}{2}x_2 + \ 5x_3 - 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2} \qquad (2.55)\\
- 4\tfrac{3}{4}x_2 + 3\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 8\tfrac{1}{4}. \qquad (2.56)
\end{aligned}$$

Considering (2.54)–(2.56) we see that the largest coefficient of x_2 occurs in (2.55). We exchange (2.55) and (2.54) to obtain

$$\begin{aligned}
4x_1 - \ 3x_2 - \ 2x_3 + \ 3x_4 &= 3 \\
- 7\tfrac{1}{2}x_2 + \ 5x_3 - 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2} \qquad (2.57)\\
- 3\tfrac{3}{4}x_2 + 4\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 5\tfrac{1}{4} \qquad (2.58)\\
- 4\tfrac{3}{4}x_2 + 3\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 8\tfrac{1}{4}. \qquad (2.59)
\end{aligned}$$

To eliminate x_2 from (2.58) and (2.59) we take multiples $\frac{1}{2}$ and $\frac{19}{30}$ of (2.57) and subtract them from (2.58) and (2.59) respectively. As a result we have the linear system

$$\begin{aligned}
4x_1 - \ 3x_2 - 2x_3 + \ 3x_4 &= 3 \\
- 7\tfrac{1}{2}x_2 + 5x_3 - \ 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2} \\
2x_3 + \ 4x_4 &= 2 \\
\tfrac{1}{3}x_3 + 4\tfrac{7}{15}x_4 &= 4\tfrac{2}{15}. \qquad (2.60)
\end{aligned}$$

No further row interchanges are necessary since the coefficient of x_3 having the largest absolute value is already in its correct place. We eliminate x_3 from (2.60) using the multiplier $\frac{1}{6}$ and as a result we have the linear system

$$\begin{aligned}
4x_1 - \ 3x_2 - 2x_3 + \ 3x_4 &= 3 \\
- 7\tfrac{1}{2}x_2 + 5x_3 - 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2} \\
2x_3 + \ 4x_4 &= 2 \\
3\tfrac{4}{5}x_4 &= 3\tfrac{4}{5}.
\end{aligned}$$

Solving by backward substitution we obtain $x_4 = 1.0$, $x_3 = -1.0$, $x_2 = -2.0$, and $x_1 = 1.0$.

Discussion

The point of this example is to illustrate the technique known as **partial pivoting**. At each stage of Gaussian elimination the pivotal equation is chosen to maximise the absolute value of the pivot. Thus the multipliers in the subsequent subtraction process are reduced (a division by the pivot is involved) so that they are all at most one in magnitude. Any rounding errors present are less likely to be magnified as they permeate the rest of the calculation.

The 4×4 example of this section was solved on a computer using an implementation of Gaussian elimination and computer arithmetic based on a 32-bit word

Table 2.2 Gaussian elimination with partial pivoting

	Solve the $n \times n$ linear system $\mathbf{Ax} = \mathbf{b}$ by Gaussian elimination with partial pivoting
1	Reduce to upper triangular form: For $i = 1, 2, \dots, n-1$
1.1	Find the j from $j = i, i+1, \dots, n$ for which $\lvert a_{ji} \rvert$ is a maximum
1.2	If $i \neq j$ exchange rows i and j (including b_i and b_j)
1.3	Subtract multiples of row i (including b_i) from rows $i+1, i+2, \dots, n$ so that the coefficient of x_i is eliminated from those rows
2	Backward substitution: a_{ij} and b_i are current values of the elements of $\mathbf{A}$ and $\mathbf{b}$ as the elimination proceeds
2.1	$x_n = b_n / a_{nn}$
2.2	For $i = n-1, n-2, \dots, 1$, $x_i = (b_i - \sum_{j=i+1}^{n} a_{ij} x_j)/a_{ii}$

(equivalent to retaining at least six significant figures). Without partial pivoting the solution obtained was $x_1 = 1.00000$, $x_2 = -2.00000$, $x_3 = -0.999999$, and $x_4 = 1.00000$, whereas elimination with pivoting gave the same solution except for $x_3 = -1.00000$. This may seem to be an insignificant improvement but the example illustrates how the method works. Even with 64-bit arithmetic it is confirmed by experience that partial pivoting particularly when used in solving larger systems of perhaps 10 or more equations is likely to be beneficial. As we will see, the form of the coefficient matrix can be crucial. For these reasons commercial implementations of Gaussian elimination always employ partial pivoting. The method is summarised in Table 2.2.

2.3.3 Multiple Right-Hand Sides

In a modelling process the right-hand side values often correspond to boundary values which may, for example, be the external loads on a structure or the resources necessary to fund various activities. As part of the process it is usual to experiment with different boundary conditions in order to achieve an optimal design. Gaussian elimination may be used to solve systems having several right-hand sides with little more effort than that involved in solving for just one set of right-hand side values.

Problem

Find three separate solutions of the system of (2.33)–(2.36) corresponding to three different right-hand sides.

Writing the system in the form $\mathbf{Ax} = \mathbf{b}$ we have three different vectors $\mathbf{b}$, namely

$$\begin{bmatrix} 8 \\ 6 \\ 3 \\ 9 \end{bmatrix}, \quad \begin{bmatrix} 2 \\ 3 \\ -1 \\ 4 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 \\ -6 \\ -3 \\ 9 \end{bmatrix}.$$

The three problems may be expressed in the following compact form

$$\begin{aligned}
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8, \quad 2, \quad 1\\
x_1 - 3x_2 + 4x_3 + 3x_4 &= 6, \quad 3, -6\\
4x_1 + 3x_2 - 2x_3 + 3x_4 &= 3, -1, -3\\
x_1 - 4x_2 + 3x_3 + 3x_4 &= 9, \quad 4, \quad 9.
\end{aligned}$$

Solution

We can solve all three equations at once by maintaining not one, but three columns on the right-hand side. The interchanges required by partial pivoting and the operations on the coefficient matrix do not depend on the right-hand side values and so need not be repeated unnecessarily. After the first exchange we have

$$\begin{aligned}
4x_1 + 3x_2 - 2x_3 + 3x_4 &= 3, -1, -3\\
x_1 - 3x_2 + 4x_3 + 3x_4 &= 6, \quad 3, -6\\
2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8, \quad 2, \quad 1\\
x_1 - 4x_2 + 3x_3 + 3x_4 &= 9, \quad 4, \quad 9
\end{aligned}$$

followed by

$$\begin{aligned}
4x_1 - \quad 3x_2 - \quad 2x_3 + \quad 3x_4 &= 3, \quad -1, \quad -3\\
-3\tfrac{3}{4}x_2 + 4\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 5\tfrac{1}{4}, \quad 3\tfrac{1}{4}, \ -5\tfrac{1}{4}\\
-7\tfrac{1}{2}x_2 + \quad 5x_3 - 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2}, \quad 2\tfrac{1}{2}, \quad 2\tfrac{1}{2}\\
-4\tfrac{3}{4}x_2 + 3\tfrac{1}{2}x_3 + 2\tfrac{1}{4}x_4 &= 8\tfrac{1}{4}, \quad 4\tfrac{1}{4}, \quad 9\tfrac{3}{4}.
\end{aligned}$$

Continuing the elimination we obtain

$$\begin{aligned}
4x_1 - \quad 3x_2 - 2x_3 + \quad 3x_4 &= 3, \quad -1, \quad -3\\
-7\tfrac{1}{2}x_2 + 5x_3 - 3\tfrac{1}{2}x_4 &= 6\tfrac{1}{2}, \quad 2\tfrac{1}{2}, \quad 2\tfrac{1}{2}\\
2x_3 + \quad 4x_4 &= 2, \qquad 2, \quad -6\tfrac{1}{2}\\
3\tfrac{4}{5}x_4 &= 3\tfrac{4}{5}, \quad 2\tfrac{1}{3}, \quad 9\tfrac{1}{4}.
\end{aligned}$$

Backward substitution is applied to each of the right-hand sides in turn to yield the required solutions, which are $(1, -2, -1, 1)^T$, $(-0.2456, -0.7719, -0.2281, 0.6190)^T$ and $(-1.4737, -6.8816, -8.1184, 2.4342)^T$ (to 4 decimal places).

Discussion

In solving a system of equations with several right-hand sides the reduction to upper triangular form has been performed once only. This is significant since the computation involved in the elimination process for larger systems is significantly greater than the computation involved in backward substitution and hence dominates the overall effort of obtaining a solution. If the system $\mathbf{Ax} = \mathbf{b}$ is to be repeatedly solved for the same $\mathbf{A}$ but for different $\mathbf{b}$ it makes sense to save and re-use the **LU** decomposition of $\mathbf{A}$.

2.4 Singular Systems

Not all systems of linear equations have a unique solution. In the case of two equations in two unknowns we may have the situation whereby the equations may be represented as parallel lines in which case there is no solution. On the other hand the representative lines may be coincident in which case every point on the lines is a solution and so we have an infinity of solutions. These ideas extend to larger systems. If there is no unique solution then there is either no solution at all or else there is an infinity of solutions and we would expect Gaussian elimination to break down at some stage.

Problem

Solve the system

$$2x_1 - 6x_2 + 4x_3 - 2x_4 = 8 \tag{2.61}$$

$$x_1 - 3x_2 + 4x_3 + 3x_4 = 6 \tag{2.62}$$

$$x_1 - x_2 + 2x_3 - 5x_4 = 3 \tag{2.63}$$

$$x_1 - 4x_2 + 3x_3 + 3x_4 = 9. \tag{2.64}$$

Solution

Applying Gaussian elimination with partial pivoting to the system we first eliminate x_1 to obtain

$$\begin{aligned} 2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\ 2x_3 + 4x_4 &= 2 \\ 2x_2 \qquad - 4x_4 &= -1 \\ -x_2 + x_3 + 4x_4 &= 5. \end{aligned}$$

Partial pivoting and elimination of x_2 gives

$$\begin{aligned} 2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\ 2x_2 \qquad - 4x_4 &= -1 \\ 2x_3 + 4x_4 &= 2 \\ x_3 + 2x_4 &= 4\tfrac{1}{2}. \end{aligned}$$

Finally, elimination of x_3 gives

$$\begin{aligned} 2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\ 2x_2 \qquad - 4x_4 &= -1 \\ 2x_3 + 4x_4 &= 2 \\ 0x_4 &= 3\tfrac{1}{2}. \end{aligned}$$

Clearly we have reached a nonsensical situation which suggests that 0 is equal to $3\frac{1}{2}$. This contradiction suggests that there is an error or a wrong assumption in the modelling process which has provided the equations.

Changing the right-hand side of (2.61)–(2.64) to $(8, 6, 3, 5.5)^T$ and applying elimination gives

$$\begin{aligned} 2x_1 - 6x_2 + 4x_3 - 2x_4 &= 8 \\ 2x_2 \qquad\quad - 4x_4 &= -1 \\ 2x_3 + 4x_4 &= 2 \\ 0x_4 &= 0 \end{aligned}$$

which is more sensible than before, if a little odd. There is no longer any contradiction. In effect we are free to choose *any* value we like for x_4 and to employ this value into the backward substitution process. In the first system we had no solution; now we have an infinity of solutions.

Discussion

It is possible to check that the rows (as vectors) of the coefficient matrix of the system (2.61)–(2.64) are linearly dependent. It follows that the columns are also linearly dependent and vice-versa. Such matrices are said to be **singular**. A system for which the coefficient matrix is singular (a **singular system**) has no unique solution. Further, if for a singular system the right-hand side as a column vector is a linear combination of the column vectors of the matrix of coefficients then there is an infinity of solutions, otherwise there are no solutions. In practice however, because of the limitations of computer arithmetic it may be difficult to determine if a system is genuinely singular or if it is just close to being singular. In any event if partial pivoting produces a zero or nearly zero pivot the validity of the model should be questioned.

2.5 Symmetric Positive Definite Systems

It is not uncommon for matrices which arise in the modelling of physical processes to be symmetric and positive definite. A matrix $\mathbf{A}$ is symmetric and positive definite if $\mathbf{A} = \mathbf{A}^T$ and $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all non-zero vectors $\mathbf{x}$. As an example the symmetric stiffness matrix which arises in finite element analysis and which may be regarded as a generalisation of the modulus of elasticity from Hooke's Law (2.1) is positive definite. We refer to a linear system in which the coefficient matrix is symmetric and positive definite as a symmetric positive definite system.

Problem

Solve the 4×4 system:

$$\begin{aligned}
16x_1 + \ \ 3x_2 + 4x_3 + 2x_4 &= 25\\
3x_1 + 12x_2 + 2x_3 - \ \ x_4 &= 16\\
4x_1 + \ \ 2x_2 + 8x_3 - \ \ x_4 &= 13\\
2x_1 - \ \ \ x_2 - \ \ x_3 + 2x_4 &= \ \ 2.
\end{aligned}$$

Solution

By Gaussian elimination we have

$$\begin{aligned}
16x_1 + \quad 3x_2 + \quad 4x_3 + \quad 2x_4 &= 25\\
11.4375x_2 + 1.25x_3 - 1.375x_4 &= 11.3125\\
1.25x_2 + \quad 7x_3 - \quad 1.5x_4 &= 6.75\\
-1.375x_2 - \quad 1.5x_3 + \ 1.75x_4 &= -1.125
\end{aligned} \tag{2.65}$$

then

$$\begin{aligned}
16x_1 + \quad x_2 + \quad 4x_3 + \quad 2x_4 &= 25\\
11.4375x_2 + \ 1.25x_3 - \ 1.375x_4 &= 11.3125\\
6.8634x_3 - 1.3497x_4 &= 5.5137\\
-1.3497x_3 + 1.5847x_4 &= 0.2350
\end{aligned} \tag{2.66}$$

and finally

$$\begin{aligned}
16x_1 + \quad 3x_2 + \quad 4x_3 + \quad 2x_4 &= 25\\
11.4375x_2 + \ 1.25x_3 - \ 1.375x_4 &= 11.3125\\
6.8634x_3 - 1.3497x_4 &= 5.5137\\
1.3913x_4 &= 1.3913.
\end{aligned}$$

Backward substitution produces $x_4 = 1$, $x_3 = 1$, $x_2 = 1$, $x_1 = 1$.

Discussion

It should be noted that at each stage the size of the pivot cannot be increased by row interchanges and so partial pivoting is not required. This is a feature of symmetric positive definite systems. Furthermore it is possible to get by with fewer calculations. For example in the first reduction the numbers 1.25, -1.375 and -1.5 need only be calculated once, whilst in the second reduction the number -1.3497 need only be calculated once. In effect it is sufficient to compute only the entries in the upper triangular portion of the coefficient matrix as it is transformed. Elements in the lower triangle are deduced by symmetry arguments. Roughly speaking, we have halved the workload and halved the storage requirements. In a large system such economies could be significant.

Although it is easy to test the symmetry of a given matrix, it is not so easy to establish if a symmetric matrix is positive definite. A symmetric matrix is positive definite if it diagonally dominant (that is if each diagonal element is larger in absolute value than the sum of the absolute values of the other elements on the same row). However being diagonally dominant is not a necessary condition for a symmetric matrix to be positive definite. Often the only real way to check is to apply Gaussian elimination. If, at any stage, partial pivoting is necessary then the matrix is not positive definite. In practice therefore when the modelling process requires the repeated solution of a symmetric linear system it is advisable to try the system on a program which assumes that the coefficient matrix is positive definite, but which flags an error if a non-positive definite condition is detected.

2.6 Iterative Refinement

The effect of rounding error in Gaussian elimination is to yield a computed solution which is a perturbation of the mathematically correct solution. Partial pivoting may help to reduce the accumulative effect of these errors, but there is no guarantee that it will eliminate them entirely. Iterative refinement is a process by which a first computed solution can sometimes be improved to yield a more accurate solution. In the following example we illustrate the method by working to limited accuracy and drawing conclusions that may be applied when using computer arithmetic.

Problem

Solve the 4×4 system

$$0.366x_1 + 0.668x_2 - 0.731x_3 - 0.878x_4 = -0.575 \tag{2.67}$$

$$0.520x_1 + 0.134x_2 - 0.330x_3 + 0.484x_4 = 0.808 \tag{2.68}$$

$$0.738x_1 - 0.826x_2 + 0.194x_3 - 0.204x_4 = -0.098 \tag{2.69}$$

$$0.382x_1 - 0.667x_2 + 0.270x_3 - 0.255x_4 = -0.270 \tag{2.70}$$

using Gaussian elimination.

Solution

The system has the exact solution $x_1 = 1$, $x_2 = 1$, $x_3 = 1$ and $x_4 = 1$. However, working to just three significant figures, Gaussian elimination with partial pivoting produces a solution $x_1 = 1.14$, $x_2 = 1.18$, $x_3 = 1.23$ and $x_4 = 0.993$. Substituting these values back into the left-hand side of the equations and working to three-figure accuracy produces -0.575, 0.808, -0.0908 and -0.270, which suggests that this computed solution is accurate. However repeating the substitution and working to six significant figures produces a slightly different set of values. Subtracting the three-figure result from the corresponding six-figure result for each equation, gives differences of −0.00950 for (2.67), −0.0176 for (2.68), −0.000688 for (2.69) and 0.00269 for (2.70). These differences are known as **residuals**. Since these residual values are small in relation to the coefficients in the equation we can suppose we are fairly close to the true solution.

More formally, if we were to assume that the true solution differs from our calculated 3-figure solution by amounts δ_1, δ_2, δ_3 and δ_4, that is the true solution is $1.14 + \delta x_1$, $1.18 + \delta_2$, $1.23 + \delta_3$, $0.993 + \delta x_4$, we would have

$$\begin{aligned}
&0.366(1.14+\delta_1)+0.668(1.18+\delta_2)\\
&\qquad -0.731(1.23+\delta_3)-0.878(0.993+\delta_4)=-0.575\\
&0.520(1.14+\delta_1)+0.134(1.18+\delta_2)\\
&\qquad -0.330(1.23+\delta_3)+0.484(0.993+\delta_4)=\quad 0.808\\
&0.738(1.14+\delta_1)-0.826(1.18+\delta_2)\\
&\qquad +0.194(1.23+\delta_3)-0.204(0.993+\delta_4)=-0.098\\
&0.382(1.14+\delta_1)-0.667(1.18+\delta_2)\\
&\qquad +0.270(1.23+\delta_3)-0.255(0.993+\delta_4)=-0.270.
\end{aligned}$$

We are now in a position to find the corrections δ_1, δ_2, δ_3 and δ_4. Using the calculations we have already performed to find the residuals we have

$$\begin{aligned}
0.366\delta_1+0.668\delta_2-0.731\delta_3-0.878\delta_4&=-0.00950\\
0.520\delta_1+0.134\delta_2-0.330\delta_3+0.484\delta_4&=-0.0176\\
0.738\delta_1-0.826\delta_2+0.194\delta_3-0.204\delta_4&=-0.000688\\
0.382\delta_1-0.667\delta_2+0.270\delta_3-0.255\delta_4&=\quad 0.00269.
\end{aligned}$$

Using 3 significant figure arithmetic again we find that Gaussian elimination with partial pivoting produces a solution $\delta_1 = -0.130$, $\delta_2 = -0.166$, $\delta_3 = -0.210$ and $\delta_4 = 0.00525$, and adding these values to our initial solution gives $x_1 = 1.01$, $x_2 = 1.01$, $x_3 = 1.02$ and $x_4 = 0.998$. Although this is still not completely accurate, it is slightly better than the previous solution.

Discussion

The example is an illustration of the technique known as **iterative refinement**. Computer calculations inevitably involve rounding errors which result from performing arithmetic to a limited accuracy. Iterative refinement is used to try to restore a little more accuracy to an existing solution. However it does depend upon being able to calculate the residuals with some accuracy and this will involve using arithmetic which is more accurate than that of the rest of the calculations. Note that the original coefficient matrix is used in finding the improved solution and this can lead to economies when applied to large systems. The method of iterative refinement as used with Gaussian elimination is summarised in Table 2.3.

Table 2.3 Gaussian elimination with iterative refinement

	Solve the $n \times n$ linear system $\mathbf{Ax} = \mathbf{b}$ by Gaussian elimination with iterative refinement
1	Set $\mathbf{s}$ to $\mathbf{0}$
2	Solve $\mathbf{Ax} = \mathbf{b}$ using Gaussian elimination with partial pivoting and add the solution to $\mathbf{s}$
3	Compute the residual $\mathbf{r} = \mathbf{b} - \mathbf{As}$ using a higher accuracy than the rest of the calculation
4	If $\mathbf{r}$ is acceptably small then stop, $\mathbf{s}$ is the solution otherwise set $\mathbf{b} = \mathbf{r}$
5	Repeat from step 2 unless $\mathbf{r}$ shows no signs of becoming acceptably small

Iterative refinement could be continued until the residuals stabilise at or very near to zero. In practice one step of iterative refinement usually suffices. If iterative refinement fails to stabilise it is likely that meaningful solutions cannot be obtained using conventional computing methods. Such systems will be discussed in the following section.

2.7 Ill-Conditioned Systems

Ill-conditioned systems are characterised by solutions which vary dramatically with small changes to the coefficients of the equations. Ill-conditioned systems pose severe problems and should be avoided at all costs. The presence of an ill-conditioned system may be detected by making slight changes to the coefficients and noticing if these produce disproportionate changes in the computed solution. An alternative method would be to apply iterative refinement as signs of non-convergence would indicate ill-conditioning. There can be no confidence in the computed solution of an ill-conditioned system, since the transfer to a computer may well introduce small distortions in the coefficients which, in addition to the limitations of computer arithmetic, is likely to produce yet further divergence from the original solution. The presence of an ill-conditioned system should be taken as a warning that the situation being modelled is either so inherently unpredictable or unstable as to defy analysis, or that the modelling process itself is at fault, perhaps through experimental or observational error not supplying sufficiently accurate information. Examples of ill-conditioned system are given in Exercises 7 and 8.

2.8 Gauss–Seidel Iteration

Systems of equations which display a regular pattern often occur in modelling physical processes for example in solving steady state temperature distributions. Linear approximations to partial derivatives over relatively small intervals are combined so that they model the differential equations over a whole surface. The resulting equations not only display a regular pattern they are also sparse in the sense that although the number of equations may be large, possibly hundreds or thousands, the number of variables in each equation is very much smaller, reflecting the way in which they have been assembled over small areas. It is perfectly possible to solve such systems using the elimination techniques considered earlier in this chapter, but that may cause storage and indexing problems. It would be possible to store a sparse coefficient matrix in a condensed form by just recording and indexing non-zero elements but the sparse quality would probably be quickly destroyed by Gaussian elimination. We consider an alternative, iterative approach to solving large sparse systems.

Problem

Solve the following system of equations using Gauss–Seidel iteration.

$$\begin{aligned}
6x_1 + x_2 \qquad\qquad\qquad\qquad &= 1\\
x_1 + 6x_2 + x_3 \qquad\qquad\qquad &= 1\\
x_2 + 6x_3 + x_4 \qquad\qquad &= 1\\
\ddots \qquad\qquad\qquad &\\
x_8 + 6x_9 + x_{10} &= 1\\
x_9 + 6x_{10} &= 1.
\end{aligned}$$

Solution

We adopt an iterative approach to obtaining a solution. We start with an initial guess at the solution and then form a series of further approximations which we hope eventually leads to a solution. We start by writing the equations as

$$x_1 = \frac{1 - x_2}{6} \tag{2.71}$$

$$x_2 = \frac{1 - x_1 - x_3}{6} \tag{2.72}$$

$$\vdots \tag{2.73}$$

$$x_9 = \frac{1 - x_7 - x_8}{6} \tag{2.74}$$

$$x_{10} = \frac{1 - x_9}{6}. \tag{2.75}$$

As an initial guess at the solution we set each of $x_1, x_2, \ldots, x_{10}$ to $\frac{1}{6}$, not an unreasonable choice since this is the solution if we ignore the off-diagonal terms in the original coefficient matrix.

We have from (2.71)

$$x_1 = \frac{1 - \frac{1}{6}}{6} = 0.1389$$

Table 2.4 Gauss–Seidel iteration, results

Step	x_1	x_2	x_3	x_4	x_5	x_6	$\cdots$	x_{10}
1	0.1667	0.1667	0.1667	0.1667	0.1667	0.1667	$\cdots$	0.1667
2	0.1389	0.1157	0.1196	0.1190	0.1191	0.1190	$\cdots$	0.1468
3	0.1474	0.1222	0.1265	0.1257	0.1259	0.1258	$\cdots$	0.1465
4	0.1463	0.1212	0.1255	0.1248	0.1249	0.1249	$\cdots$	0.1464
5	0.1465	0.1213	0.1256	0.1249	0.1250	0.1250	$\cdots$	0.1464
6	0.1464	0.1213	0.1256	0.1249	0.1250	0.1250	$\cdots$	0.1464
7	0.1464	0.1213	0.1256	0.1249	0.1250	0.1250	$\cdots$	0.1464

Table 2.5 Gauss–Seidel iteration

	Solve the $n \times n$ sparse linear system $\mathbf{Ax} = \mathbf{b}$ by Gauss–Seidel iteration
1	Set $\mathbf{x}$ to an estimate of the solution
2	For $i = 1, 2, \ldots, n$ evaluate $x_i = (b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j)/a_{ii}$ using the most recently found values of x_j
3	Repeat from step 2 until the values x_i settle down. If this does not happen abandon the scheme

and so from (2.72)

$$x_2 = \frac{1 - 0.1389 - \frac{1}{6}}{6} = 0.1157$$

and we find values for $x_3, x_4, \ldots, x_{10}$ in a similar manner to give the values shown in the first line of Table 2.4. Using these as new values for the right-hand sides, in the second iteration we obtain

$$x_1 = \frac{1 - 0.1157}{6} = 0.1474 \tag{2.76}$$

and so from (2.72)

$$x_2 = \frac{1 - 0.1474 - 0.1196}{6} = 0.1222 \tag{2.77}$$

and we find values for $x_3, x_4, \ldots, x_{10}$ in a similar manner to give the values shown in the second line of Table 2.4. After further iterations the values $x_1, x_2, \ldots x_{10}$ converge to 0.1464, 0.1213, 0.1256, 0.1249, 0.1250, 0.1250, 0.1249, 0.1256, 0.1213 and 0.1464 respectively. A calculation of residuals would confirm that we have a solution to the equations. The process is known as **Gauss–Seidel iteration**. The method is summarised in Table 2.5 in which it is assumed coefficients are stored in a conventional matrix, although in practice coefficients and variables would not necessarily be stored in full matrix or vector form.

Discussion

Unfortunately it can be readily demonstrated that Gauss–Seidel iteration does not always converge (see Exercise 9). For systems in which the coefficient matrix is diagonally dominant (page 35), then Gauss–Seidel iteration is guaranteed to converge. This is not to say that non-diagonal systems cannot be solved in this way, it is just that convergence cannot be guaranteed. For this reason the method would in general only be applied to large systems, which might otherwise cause the storage problems already mentioned.

Summary In this chapter we examined what is meant by a linear relationship and how such a relationship is represented by a linear equation. We looked at Gaussian elimination for solving systems of such equations and noted that

- Gaussian elimination is essentially a reduction to upper triangular form followed by backward substitution.
- re-arranging the order in which the equations are presented can affect the accuracy of the computed solution, led us to the strategy known as partial pivoting.

We also considered special types of linear systems, namely

- singular systems, for which there is no unique solution and which may be identified by a failure in Gaussian elimination even when partial pivoting is included.
- symmetric positive definite systems, for which we mentioned that more efficient methods may be used.
- ill-conditioned systems, that are likely to cause problems no matter what computational methods are used.
- sparse systems for which the Gauss–Seidel iterative method may be used if computer time and storage is at a premium.

Mindful of the errors inherent in computer arithmetic we looked at ways of improving an existing solution using

- iterative refinement.

although we pointed out that in the case of ill-conditioned systems it might make matters worse.

Exercises

1. A baker has 2.5 kilos of flour, 3.5 kilos of sugar and 5.3 kilos of fruit with which to make fruit pies. The baker has utensils to make pies in three sizes—small, medium and large. Small pies require 30 grams of flour, 20 grams of sugar and 40 grams of fruit. Similarly the requirement for medium size pies is 40, 30 and 60

grams respectively, and for large pies 60, 50 and 90 grams. The baker wishes to use up all the ingredients. Formulate a linear system to decide if this is possible.

Let the number of small, medium and large pies to be made be x, y and z. Taking into account that 1 kilo = 1000 grams we have

$$\begin{aligned} 3x + 2y + 4z &= 250 \quad \text{(flour to be used)} \\ 4x + 3y + 6z &= 350 \quad \text{(sugar to be used)} \\ 6x + 5y + 9z &= 530 \quad \text{(fruit to be used).} \end{aligned}$$

If the solution to these equations turns out to consist of whole numbers then the baker can use up all the ingredients. If some or all of x, y and z were not whole numbers then the baker is not able to use up all the ingredients, deciding how many pies of each size to make in order to minimise the quantity of ingredients left over is the type of problem we look at in Chap. 7.

Assuming the equations are written in the form $\mathbf{Ax} = \mathbf{b}$ use the following code to find a solution. The code uses the left-division operator, \ as described in Sect. 1.4.

```
A = [ 3  2  4;  4  3  6;  6  5  9 ]
b = [ 250  350  530 ] '
x = A\b
```

2. (i) Reduce the following linear system to upper triangular form.

$$\begin{aligned} 2x_1 - x_2 - 4x_3 &= 5 \\ x_1 + x_2 + 2x_3 &= 0 \\ 6x_1 + 3x_2 - x_3 &= -2.5. \end{aligned}$$

Form the matrix of coefficients of the upper triangular system in the matrix **U** and the corresponding right-side in the vector **r**. Use the following Matlab code to make copies of **A** and **b** in **U** and **r** respectively. A multiple (the pivot) of row 1 is subtracted from row 2 and row 3, then a multiple (another pivot) of row 2 is subtracted from row 3.

```
U = A;
r = b;
for row = 1 : 2;
  for i = row + 1 : 3;
    pivot=U(i, row)/U(row, row)
    U(i, :) = U (i, :) − pivot * U(row, :)
    r(i) = r(i) − pivot∗ r(row)
  end
end
```

As something of a programming challenge try modifying the code to handle larger matrices and to allow partial pivoting and deal with zero pivots.

(ii) Take the solution from part (i) and apply backward substitution to find a solution to the original linear system. Use the following Matlab code, which assumes that the coefficients of the upper triangular system are stored in the matrix **U** and that the modified right hands are stored in **r**.

```
x(3) = r(3)/U(3, 3);
for i = 2 : −1 : 1
  sum = 0;
  for j = i+1 : 3 ;
    sum = sum + U(i, j)*x(j);
  end;
  x(i) = (r(i) − sum)/U (i, i);
end;
x
```

Check the validity of the solution by evaluating $\mathbf{U} * \mathbf{x}$, where **x** is the solution as a column vector. All being well this should produce the original **b**.

3. Matlab has a function *lu* which provides the **LU** factorisation of a matrix. Form the matrix, **A** of coefficients of the following system in a Matlab program.

$$\begin{aligned} x_1 + 2x_2 + 3x_3 + \ x_4 &= \ \ 8 \\ 2x_1 + 4x_2 - \ x_3 + \ x_4 &= \ \ 1 \\ x_1 + 3x_2 - \ x_3 + 2x_4 &= \ \ 1 \\ -3x_1 - \ x_2 - 3x_3 + 2x_4 &= -7. \end{aligned}$$

Use the *lu* command in the form [L U] =*lu*(A), which returns an upper triangular **U** and a (possibly) permuted lower triangular **L** so that $\mathbf{Ax} = \mathbf{LUb}$, where **b** is the column vector of right-sides.

Having established matrices **L** and **U** we have $\mathbf{LUx} = \mathbf{b}$. Writing $\mathbf{Ly} = \mathbf{b}$, find **y** by using the left-division operator. We now have $\mathbf{Ux} = \mathbf{y}$. Find **x** by using the left-division operator. Check the result by comparing **Ax** and **b**.

Alternatively use the *lu* command in the form [L U P] = *lu*(A) so that **U** is not permuted and a permutation matrix **P** is returned. In this case **b** would be replaced by **Pb** in the scheme shown above, and so **x** could be found by left-division.

4. As an example of how rounding error can affect the solutions to even relatively small systems solve the following equations using the Matlab left-division \ operator.

$$\begin{array}{rrrrl} -0.6210x_1 & 0.0956x_2 & -0.0445x_3 & 0.8747x_4 = & 5.3814 \\ 0.4328x_1 & -0.0624x_2 & 8.8101x_3 & -1.0393x_4 = & -1.0393 \\ -0.0004x_1 & -0.0621x_2 & 5.3852x_3 & -0.3897x_4 = & -0.3897 \\ 3.6066x_1 & 0.6536x_2 & 0.8460x_3 & -0.2000x_4 = & -0.0005 \end{array}$$

Use both 64-bit precision (the default option) and then use 32-bit precision by converting the variables using the *single* operator, for example $B = single(A)$. For both cases set *format long* to show the maximum number of decimal places for the chosen word length.

5. Form an **LU** factorisation of the following symmetric matrix to show that it is not positive definite.

$$\begin{pmatrix} 4 & 1 & -1 & 2 \\ 1 & 3 & -2 & -1 \\ -1 & -2 & 1 & 6 \\ 2 & -1 & 6 & 1 \end{pmatrix}$$

   Using a little ingenuity we can find a non-zero vector such as $\mathbf{x}^T = (0 \;\; 1 \;\; 1 \;\; 0)$ that does not satisfy the requirement $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$.

6. Show that the following system has no solution.

$$\begin{aligned} 2x_1 + 3x_2 + 4x_3 &= 2 \\ x_1 - 2x_2 + 3x_3 &= 5 \\ 3x_1 + \;\; x_2 + 7x_3 &= 8 \end{aligned}$$

   Use the LU factorisation as provided by the function *lu* to show that the matrix of coefficients is singular. Show by algebraic manipulation that the system does not have a unique solution. Decide if there is no solution or an infinity of solutions.

7. Solve the following systems of linear equations, one of which is a slight variation of the other to show that we have a example of an ill-conditioning

$$\begin{array}{rr} 0.9740x_1\ 0.7900x_2\ 0.3110x_3 = 1.0 & 0.9736x_1\ 0.7900x_2\ 0.3110x_3 = 1.0 \\ -0.6310x_1\ 0.4700x_2\ 0.2510x_3 = 0.5 & -0.6310x_1\ 0.4700x_2\ 0.2506x_3 = 0.5 \\ 0.4550x_1\ 0.9750x_2\ 0.4250x_3 = 0.75 & 0.4550x_1\ 0.9746x_2\ 0.4250x_3 = 0.75. \end{array}$$

8. As an example of severe ill-conditioning consider the Hilbert[2] matrix $\mathbf{H}_n$, which has the form

$$\mathbf{H}_n = \begin{pmatrix} 1 & 1/2 & 1/3 & 1/4 & \dots & 1/n \\ 1/2 & 1/3 & 1/4 & 1/5 & \dots & 1/(n+1) \\ 1/3 & 1/4 & 1/5 & 1/6 & \dots & \\ 1/4 & 1/5 & 1/6 & 1/7 & \dots & \\ \vdots & & & & & \vdots \\ 1/n & 1/(n+1) & \dots & & & 1/(2n-1) \end{pmatrix}.$$

   Use the Matlab command $x = ones(n, 1)$ to generate a column vector $\mathbf{x}$ of n 1's and the command $H = hilb(n)$ to generate the Hilbert matrix H_n of order n. Form the vector $\mathbf{b}_n = \mathbf{H}_n\mathbf{x}$.

   Solve the system $\mathbf{H}_n\mathbf{x} = \mathbf{b}_n$ for various n using the left division operator. In theory $\mathbf{x}$ should be $(1, \dots, 1)$, but see what happens. Begin with $n = 3$ increase the value with suitable increments up to around $n = 18$ to show the effects of ill-conditioning and rounding error. Iterative refinement will not be of any help in this case.

[2]David Hilbert, 1862–1943. In 1900 he put forward a list of 23 important unsolved problems some of which remain unsolved.

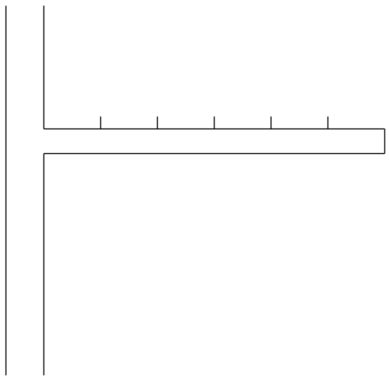


Fig. 2.2 Branch of a tree

9. Write a Matlab program to verify the results quoted for the Gauss–Seidel problem of Sect. 2.8. Use the program to show that Gauss–Seidel iteration applied to the following system with initial estimates of $x_1, x_2, \ldots, x_{10} = 0.5$ (or other estimates which appear close to the solution) does not converge.

$$\begin{aligned}
\tfrac{1}{6}x_1 + x_2 &= 1\\
x_1 + \tfrac{1}{6}x_2 + x_3 &= 1\\
x_2 + \tfrac{1}{6}x_3 + x_4 &= 1\\
&\vdots\\
x_8 + \tfrac{1}{6}x_9 + x_{10} &= 1\\
x_9 + \tfrac{1}{6}x_{10} &= 1.
\end{aligned}$$

10. A bird lands on the branch of a tree and takes steps to the left or to the right in a random manner. If the bird reaches the end of the branch or the tree trunk it flies away. Assume that the branch has the form shown in Fig. 2.2, namely that it has a length equal to six steps. What are the probabilities of the bird reaching the end of the branch from any of the five positions shown in Fig. 2.2. Let the probability of the bird reaching the end of the branch from a position i steps from the trunk be P_i, $i = 0, 1, \ldots, 6$. $P_0 = 0$ since if the bird is at the tree trunk it flies away, equally $P_6 = 1$ since the bird is at the end of the branch. For any other position we have $P_i = \frac{1}{2}(P_{i+1} + P_{i-1})$ for $i = 1, 2, \ldots, 5$.

    This gives rise to the equations

$$\begin{aligned}
P_1 &= \tfrac{1}{2}P_2\\
P_2 &= \tfrac{1}{2}(P_3 + P_1)\\
P_3 &= \tfrac{1}{2}(P_4 + P_2)\\
P_4 &= \tfrac{1}{2}(P_5 + P_3)\\
P_5 &= \tfrac{1}{2}(1 + P_4).
\end{aligned}$$

Set up a Gauss–Seidel iterative scheme in Matlab to solve the problem. Choose initial estimates such as $P_i = 0$, $i = 1, 2, \ldots, 5$. Confirm that the results are in line with expectations.

Chapter 3
Nonlinear Equations

Aims In this chapter we look at a variety of methods that can be used to find a *root* (or *zero*, the terms are interchangeable) of a function $f(x)$ or a single variable x, which is equivalent to solving the equation $f(x)=0$. We will also look at a method for solving systems of nonlinear equations and so we extend the scope of Chap. 2 which only considered systems of linear equations.

We look at the following root-finding methods for a single nonlinear equation

- the bisection method.
- false position.
- the secant method.
- the Newton–Raphson[1] method.
- Newton's method for a system of nonlinear equations.

It should be noted that in general a nonlinear equation will have a number of roots; for example, if f is polynomial of degree n in x then it has n roots, some of which may be multiple roots. Further, some, or all, of the roots of a polynomial equation may be complex. The methods we investigate are designed to locate an individual real root of a function $f(x)$ within a given range of x values.

Overview The problems involved in finding a root of an equation which models a nonlinear relationship are more complex than those for linear equations. Indeed it is not possible to provide a method which can be guaranteed to work for every single case. For this reason it is as well to be acquainted with more than one method and to be aware of the strengths and weaknesses of particular methods through theoretical insight and practical experience. The four methods for a single nonlinear equation are all of the same basic, iterative, form. Starting with one or more approximations to a root of the equation $f(x)=0$ they generate a sequence that may or may not converge to a root. The differences lie in

[1]Joseph Raphson, 1648–1715, English mathematician credited with the independent discovery of Newton's method for solving equations numerically.

- the number of starting values.
- the guarantee, or otherwise, of convergence.
- the speed of convergence.
- the cost, in terms of the work to be done per iteration.

Which of the possibly many roots is located by the methods will depend on the choice of starting position.

Acquired Skills After reading this chapter you will understand the principles underlying iterative root-finding methods. You will know how to compare the theoretical properties of root-finding methods and what these properties mean in practice. Using this knowledge you will be able to choose an appropriate root-finding method for the problem at hand.

3.1 Introduction

As an example of a nonlinear equation we consider the Butler–Volmer equation which in electrochemical processes relates current density to potential. The significance of these technical terms need not concern us. For our purposes the Butler–Volmer equation may be expressed as

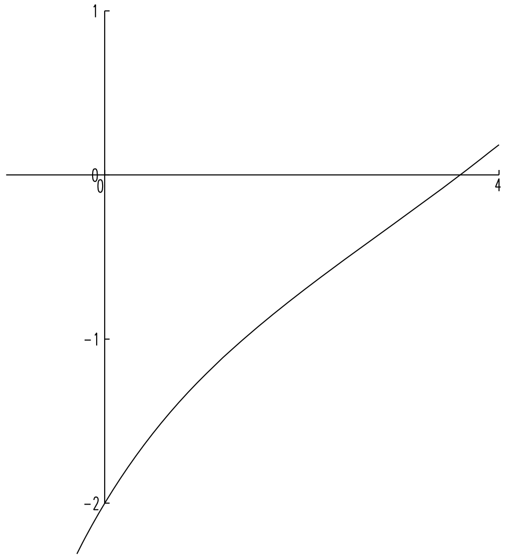
$$f(x) = e^{\alpha x} - e^{-(1-\alpha)x} - \beta. \tag{3.1}$$

In particular we solve

$$f(x) = 0 \quad \text{where } f(x) = e^{0.2x} - e^{-0.8x} - 2. \tag{3.2}$$

A graph of this function for x for $0 \le x \le 4$ is shown in Fig. 3.1.

Fig. 3.1 Butler–Volmer function



3.2 Bisection Method

The bisection method is the simplest of all the methods for finding a root of a nonlinear equation. We start with an interval containing a root and divide it into a left and a right half. We decide which of the two halves contains a root and proceed with a further division of that half. We do this repeatedly until the interval containing a root is sufficiently narrowed down to meet the level of accuracy required. If we take the mid-point of the latest interval to be an approximation to a root, we can say that this is accurate to within ± half the width of the interval.

Problem

Use the method of bisection to find a root of the equation

$$f(x) = 0 \quad \text{where } f(x) = e^{0.2x} - e^{-0.8x} - 2.$$

Solution

From Fig. 3.1 we know that there is a single root of the equation in the interval $[0, 4]$. $f(0)$ is negative and $f(4)$ is positive, and since the function is continuous, as x increases from 0 to 4 at some point f must cross over from being negative to positive and have the value 0.

At the mid-point of the interval, where $x = 2$, we find that $f(2) = -0.71007$. It follows that there must be a root of the function to the right of $x = 2$ since $f(2)$ is negative and $f(4)$ is positive. That is, having started with the interval $[0, 4]$ we now know that there is a root of the function in the reduced interval $[2, 4]$.

We now look at $[2, 4]$ for which the mid-point is $x = 3$. Since $f(3)$ is -0.26860, which is negative, we know there must be a root of the function in the further reduced interval $[3, 4]$. Continuing in a similar manner we produce the sequence of intervals shown in Table 3.1, each of which is known to contain a root of the equation.

Table 3.1 Sequence of intervals obtained by bisection

a	b	$\frac{a+b}{2}$	$f(a)$	$f(\frac{a+b}{2})$	$f(b)$
0	4	2	−2	−0.71007	0.18478
2	4	3	−0.71007	−0.26860	0.18478
3	4	3.5	−0.26860	−0.04706	0.18478
3.5	4	3.75	−0.04706	0.06721	0.18478
3.5	3.75	3.625	−0.04706	0.00971	0.06721
3.5	3.75	3.625	−0.04706	0.00971	0.06721
3.5	3.625	3.5625	−0.04706	−0.01876	0.00971
3.5625	3.5625	3.59375	−0.01876	−0.00455	0.00971
3.59375	3.625	3.60938	−0.00455	0.00257	0.00971

Table 3.2 The bisection method

	Solve $f(x) = 0$ by bisection
1	Find an interval $[a, b]$ in which $f(x) = 0$
2	Set x^* to $\frac{a+b}{2}$
3	Calculate $f(x^*)$
4	Test for convergence
5	If $f(x^*)$ and $f(b)$ have opposite sign set a to x^*, otherwise set b to x^*
6	Repeat from step 2

Discussion

We have to decide when to stop the procedure and this will depend upon the problem under consideration. In terms of absolute values we can deduce from Table 3.1 that we have a range of x values $[3.59375, 3.60938]$ for which f is to within ± 0.005 of zero. It is important to distinguish between **absolute error**, which we have just described and **relative error** which measures relative values. In this example the relative values might be measured as the ratios $\frac{a-b}{a}$ and $\frac{f(a)-f(b)}{f(a)}$. In practice the level of acceptable error would be decided in advance.

Furthermore in considering finding a root of a function $f(x)$ we have to take into account a possible imbalance between x and $f(x)$ values. If for example the function has a very steep slope at the root, the range of x values producing a function values close to zero could be of a much smaller order of magnitude. Conversely, if the function is very flat in the region of a root the range of possible x values would be comparatively large. The bisection method allowing for a choice of absolute or relative error checking and criteria based on either x and/or $f(x)$ values as the test for convergence is summarised in Table 3.2.

3.2.1 Finding an Interval Containing a Root

In the case of the function already considered we were fortunate to have available a graph which could be used to determine an initial interval containing a root. However if such a graph is not readily available it would be advisable to generate a series of x and f values to gain a feel for the function and the presence of roots and enclosing intervals. In the event of a function having multiple roots care should be taken to choose an interval containing just one root, otherwise the bisection method (and the other methods to be considered) may yield a value which is inappropriate to the problem being modelled.

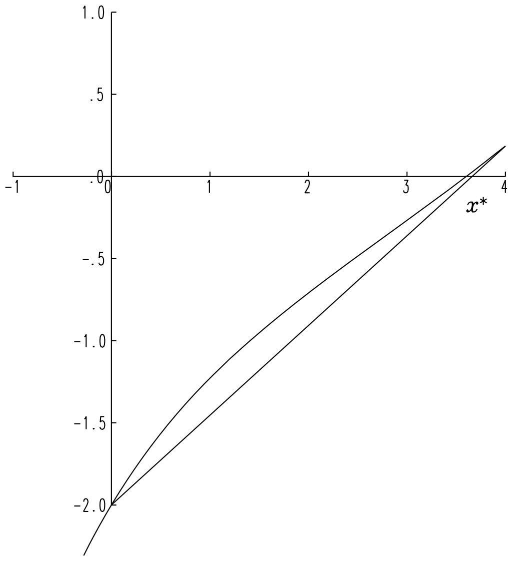


Fig. 3.2 Rule of false position, approximation to the root

3.3 Rule of False Position

The rule of false position is a version of the bisection method that takes a more informed approach to reducing the interval containing a root. It is likely therefore to require a smaller number of iterations.

We reconsider the problem of finding a root of equation

$$f(x) = 0 \quad \text{where } f(x) = e^{0.2x} - e^{-0.8x} - 2$$

in the interval $[0, 4]$. From Fig. 3.1 it is clear that there is a root closer to 4 than 0. So rather than simply take the mid-point in order to reduce the interval, we look at a point which is weighted more towards the likely position of the root. Using Fig. 3.2 we choose the point marked x^* to be the next approximation to the root rather than the mid-point. x^* is the intersection of the straight line (known as the secant) joining the points $(x = 0, y = -2)$ and $(x = 4, y = 0.18478)$. By similar triangles

$$\frac{x^*}{2} = \frac{4 - x^*}{0.18478} \tag{3.3}$$

and so

$$x^* = 3.66170. \tag{3.4}$$

Now $f(3.66170) = 0.02651$ which is positive and so we deduce that there must be a root in the interval $[0, 3.66170]$. In this reduced interval we construct a secant joining the points $(x = 0, y = -2)$ and $(x = 3.66170, y = 0.02651)$ and find by similar triangles that this cuts the x-axis at a new x^* given by

$$\frac{x^*}{2} = \frac{3.66170 - x^*}{0.02651} \tag{3.5}$$

Table 3.3 Results using the rule of false position

a	b	x^*	$f(a)$	$f(x^*)$	$f(b)$
0	4	3.66170	−2.0	0.02651	0.18478
0	3.66170	3.61380	−2.0	0.00459	0.02651

Table 3.4 The rule of false position

	Solve $f(x) = 0$ by the rule of false position
1	Find an interval $[a, b]$ in which $f(x) = 0$
2	Set x^* to $b - f(b)\frac{a-b}{f(a)-f(b)}$
3	Calculate $f(x^*)$
4	Test for convergence
5	If $f(x^*)$ and $f(b)$ have opposite sign set a to x^*, otherwise set b to x^*
6	Repeat from step 2

and so

$$x^* = 3.61380. \tag{3.6}$$

If, as before, the aim is to find an x such that $f(x)$ is within ± 0.01 of zero we are already there since $f(3.61380) = 0.00459$. The results are summarised in Table 3.3. In the more general case where we enclose a root of the function $f(x)$ in the interval $[a, b]$ we can establish that the secant joining the points $(a, f(a))$, $(b, f(b))$ cuts the x-axis at the point x^* where

$$x^* = b - f(b)\frac{a-b}{f(a) - f(b)}. \tag{3.7}$$

As in the example, the result is established using similar triangles. Although the rule of false position produces a root in fewer steps than bisection it does not necessarily produce the relatively small interval containing the root since the progression to the root may be from one side of the interval. The rule of false position is summarised in Table 3.4.

3.4 The Secant Method

The secant method uses the same basic formula as the rule of false position for finding a new approximation to the root based on two existing approximations. However the secant method does not require an initial interval containing the root and to this extent may be more useful. We reconsider the equation

$$f(x) = 0 \quad \text{where } f(x) = e^{0.2x} - e^{-0.8x} - 2$$

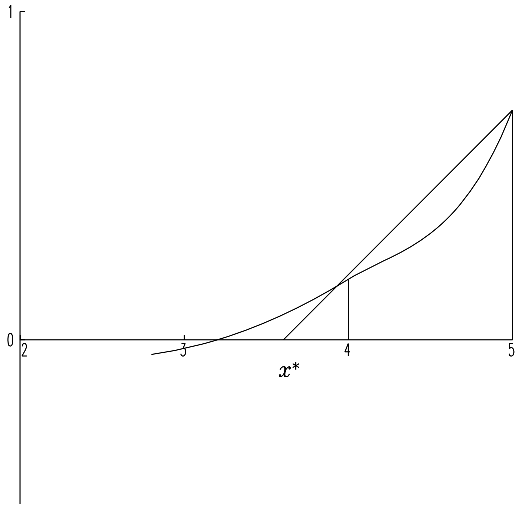


Fig. 3.3 Secant method, approximation to the root

to illustrate the method by choosing two arbitrary values from which to proceed. For no particular reason we choose $x = 5$ as the first point and $x = 4$ as the second point. We note that initial function values need not be of opposite sign. We have $f(5) = 0.69997$ and $f(4) = 0.18478$. On the basis of Fig. 3.3 the point marked x^* is likely to be a better approximation to the root.

By similar triangles we have

$$\frac{4 - x^*}{0.18478} = \frac{5 - x^*}{0.69997} \tag{3.8}$$

and so

$$x^* = 3.64134. \tag{3.9}$$

Now $f(3.64134) = 0.01718$ and so we are getting closer to the root. We repeat the method by discarding the oldest estimate (in this case, $x = 5$) and retaining the two latest estimates ($x = 4$ and $x = 3.64134$) and then find another, x^* given by

$$\frac{3.64134 - x^*}{0.01718} = \frac{4 - x^*}{0.18478} \tag{3.10}$$

and so

$$x^* = 3.60457. \tag{3.11}$$

If, as before, the aim is to find an x such that $f(x)$ is within ± 0.01 of root we are already there since $f(3.60457) = 0.00038$. To carry the method one stage further we discard $x = 4$ and retain $x = 3.64134$ and $x = 3.60457$. The next estimate is $x^* = 3.60373$ for which $f(3.60373) = 0.0$ (to 6 decimal places). The sequence is shown in Table 3.5. On the evidence presented here, the secant method converges to a root more quickly than either bisection or the rule of false position. Moreover initial estimates that are wildly inaccurate may still provide a root. A typical progression

Table 3.5 Results using the secant method

a	b	x^*	$f(a)$	$f(b)$	$f(x^*)$
5	4	3.64134	0.69997	0.18478	0.01718
4	3.64134	3.60457	0.18478	0.01718	0.00038
3.64134	3.60457	3.60373	0.01718	0.00038	0.00000

Table 3.6 The secant method

	Solve $f(x) = 0$ by the secant method
1	Choose two values a and b
2	Set x^* to $b - f(b)\frac{a-b}{f(a)-f(b)}$
3	Calculate $f(x^*)$
4	Test for convergence
5	Set a to b and b to x^*
6	Repeat from step 2

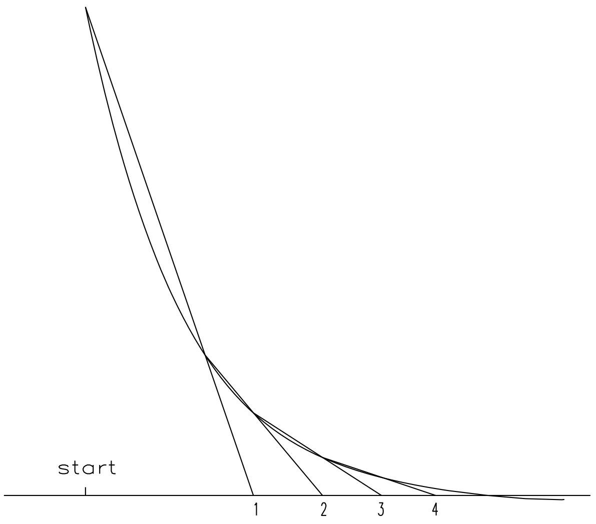


Fig. 3.4 A typical progression of secants

of secants indicating the location of a root is shown in Fig. 3.4 and the method is summarised in Table 3.6. However the method is not guaranteed to converge towards a root from any pair of starting values and will break down with a potential division by zero if any secant is parallel to the x-axis, as shown in Fig. 3.5. This latter difficulty would be overcome by temporarily reverting to the bisection method.

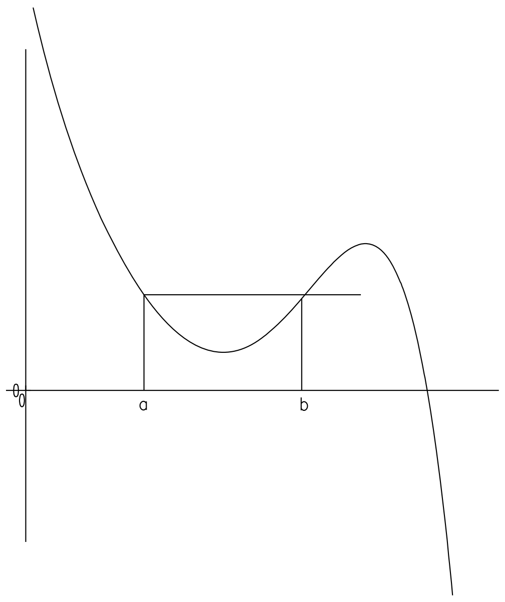


Fig. 3.5 Secant method breaks down

3.5 Newton–Raphson Method

A feature of the secant method is its attempt to follow the graph of the function using a straight line. The Newton–Raphson method pursues the idea by allowing the two points of the secant method to merge into one. In so doing it uses a single point and the tangent to the function at that point to construct a new estimate to the root. Once again we reconsider the equation

$$f(x) = 0 \quad \text{where } f(x) = e^{0.2x} - e^{-0.8x} - 2$$

to illustrate the method.

Since we have available an analytic form for f which involves simple exponentials we can readily form an analytic expression for the gradient $f'(x)$ which can therefore be used to calculate the tangent to f. We have

$$f'(x) = 0.2e^{0.2x} + 0.8e^{-0.8x}. \tag{3.12}$$

We start with the initial (arbitrary) root estimate $x = 4$, for which $f(4) = 0.18480$. From Fig. 3.6 it seems likely that x^*, the point at which the tangent to f at $x = 4$ crosses the x axis, will be a closer approximation to the root and this is indeed the case. By evaluating (3.12) at $x = 4$ we find that the gradient of the tangent at this point is 0.47772. This value is equal to the tangent of the angle to the x-axis and so

$$0.47772 = \frac{0.1848}{4 - x^*}$$

and therefore $x^* = 3.61321$.

Since $f(3.61321) = 0.00432$ we have already arrived at a value of x which gives f to within ± 0.01 of root. Generalising the argument it can be shown that if the

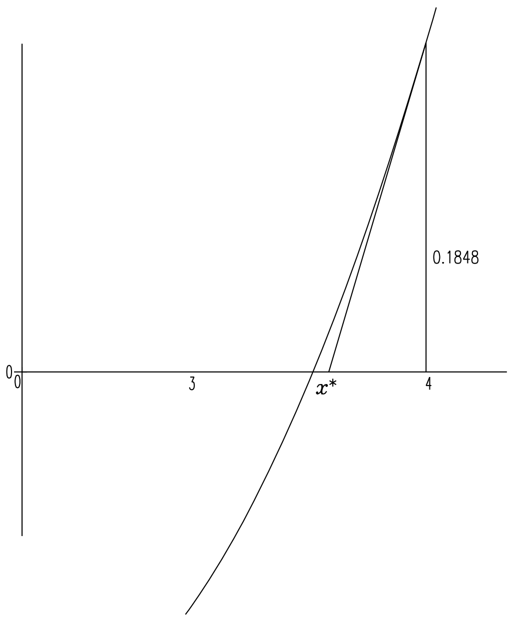


Fig. 3.6 The Newton–Raphson method

Table 3.7 The Newton–Raphson method

	Solve $f(x) = 0$ by the Newton–Raphson method
1	Choose a starting point a
2	If $f'(a) \neq 0$ replace a by $a - \frac{f(a)}{f'(a)}$ otherwise restart using a different a
3	Test for convergence
4	Repeat from step 2

tangent at a to $f(x)$ intercepts the x-axis it does so at the point

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

This is the basis of the Newton–Raphson method which is summarised in Table 3.7.

Problem

Use the Newton–Raphson method to find a root of the polynomial equation

$$f(x) = 2x^3 - 9x^2 + 12x - 6.$$

Solution

We have $f'(x) = 6x^2 - 18x + 12$. If we start at the point $x = 0$, we have $f(0) = -6$ and $f'(0) = 12$, from which it follows from a graph similar to that of Fig. 3.6 that

Table 3.8 Results using the Newton–Raphson method

a	$f(a)$	$f'(a)$	$a - \frac{f(a)}{f'(a)}$
0	−6	12	0.5
0.5	−2	4.5	0.94444
0.94444	−1.00960	0.135185	3.81384
3.81384	19.8	30.6	3.16710
3.16710	5.26	15.2	2.82010
2.82010	1.12	8.96	2.69495
2.69495	0.12014	7.07	2.67795
2.67795	0.00206	6.83	2.67765
2.67765	0.00001	6.82	

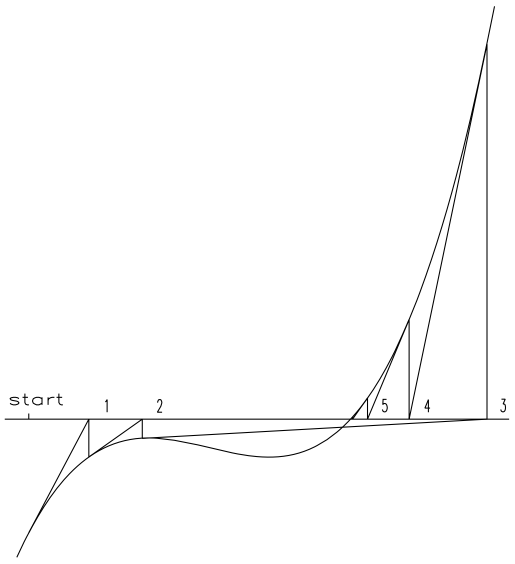


Fig. 3.7 A typical progression of tangents

the next estimate x^* of the root is calculated from

$$12 = \frac{6}{x^*} \tag{3.13}$$

and therefore $x^* = 0.5$. The method is repeated from the point $x = 0.5$ until $f(x)$ is sufficiently small. The sequence is shown numerically in Table 3.8. The progression of tangents indicating the location of the root is shown in Fig. 3.7. Notice how in the final stages the method converges very quickly.

Discussion

It is a feature of the Newton–Raphson method that if it does work it is faster to converge than any of the methods we have considered so far. The effort involved in calculating the derivative of the function is worthwhile if as in a real-time application, a rapid method is required. However, to ensure convergence it may be necessary to make a good initial estimate of the root. There are two problems which may arise using Newton–Raphson. It is possible that at some point in the sequence the tangent could be parallel to the axis, the derivative would be zero and so the formula would not be valid. Theoretically the method may cycle, the sequence of points approximating to the root may return to some earlier point but in practice that would be a very rare occurrence. In either case progress could be made by temporarily reverting to one of the former methods or by varying the current estimate in order to make progress.

3.6 Comparison of Methods for a Single Equation

We have covered a number of methods for finding a root of a function namely, bisection, rule of false position, secant and Newton–Raphson. In making comparisons between these various schemes it is necessary to take into account

1. the requirement to provide an initial interval
2. the guarantee (or otherwise) that the method will find a root
3. the relative rates at which the methods converge to a solution.

A method might be inappropriate if it requires an inordinate amount of computing time to reach a solution, particularly in *real time* applications where a rapid response might be required. The term **order of convergence** gives an indication of the rate at which an iterative process reaches a solution. The order of convergence is a measure of how the error (the absolute value of the difference between the approximation and the true solution) decreases as the iteration proceeds. For example, an iterative method having linear convergence (or order of convergence 1) might produce a sequence of errors 0.1, 0.5, 0.25, ..., that is, the error is halved at each iteration, whereas a method having quadratic convergence (or order of convergence 2) might produce a sequence of errors 0.1, 0.01, 0.001, ... that is, the error is squared at each iteration. It follows that a method having quadratic convergence generally converges much faster than a method having linear convergence. The advantages and disadvantages of each method are summarised in Table 3.9.

Table 3.9 Comparison of methods

Method	Initial interval required?	Guaranteed to find a root?	Order of convergence
Bisection	Yes	Yes	1
False position	Yes	Yes	between 1 and 1.62
Secant	No	No	1.62
Newton–Raphson	No	Yes—if initial estimate is sufficiently accurate	2

3.7 Newton's Method for Systems of Nonlinear Equations

To take matters further we consider nonlinear relationships involving more than one variable. We restrict our attention to systems of equations having the same number of equations as unknowns. Systems having an unequal number of equations and unknowns can be dealt with as an optimisation problem (Chap. 8). We write the system as

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

By way of an example we consider the following problem.

Problem

The stress distribution within a metal plate under a uniformly distributed load is modelled by (3.14) and (3.15). The variables x_1 and x_2 define the position of a point within the plate, but the precise details need not concern us.

$$f_1(x_1, x_2) = \cos(2x_1) - \cos(2x_2) - 0.4 \tag{3.14}$$

$$f_2(x_1, x_2) = 2(x_2 - x_1) + \sin(2x_2) - \sin(2x_1) - 1.2. \tag{3.15}$$

The problem is to find values for x_1 and x_2 such that $f_1(x_1, x_2) = f_2(x_1, x_2) = 0$. Although it is possible to extend the secant method to deal with such systems by using planes in 3-dimensional space rather than secants in 2-dimensional space it is easier to extend the principles of the Newton–Raphson method into what is known as **Newton's method**.

Solution

As usual we start from some initial approximation to the roots. In this case we choose $x_1 = 0.1$ and $x_2 = 0.5$. These are values which, given a familiarity with the problem being modelled, are not unreasonable. In any event $f_1(0.1, 0.5) \approx 0.04$ and $f_2 \approx 0.2$ and are therefore reasonably close to zero. In dealing with systems of nonlinear equations it is as well to exercise some care in choosing the initial approximations since in general the methods for nonlinear systems are not as robust as those for single equations and may not work for initial approximations too far from the solution. Given the initial points $x_1 = 0.1$ and $x_2 = 0.5$ we look for a better approximation x_1^* and x_2^* to make $f_1(x_1^*, x_2^*)$ and $f_2(x_1^*, x_2^*)$ closer to the root.

If the function f_1 is plotted on the axes shown in Fig. 3.8 we have a 3-dimensional shape at which the point marked A has coordinates $(0.1, 0.5)$ in the (x_1, x_2) plane. We attempt to follow the shape of the function f_1 in the $x_1 = 0.1$ plane by drawing the tangent AB to f_1 at $(0.1, 0.5)$. Similarly, we follow the shape in the $x_2 = 0.5$ plane by drawing the tangent AC to f_1 at $(0.1, 0.5)$. Since we are dealing with a solid shape, there are many tangents at any particular point. As yet the points B and C are undefined but we aim to construct a point D in the plane of $f_1 = 0$ such that CD is parallel to AB, and AC, FG and BD are parallel. The shape ABCD is our approximation to f_1 even though at this stage the point D is not uniquely defined.

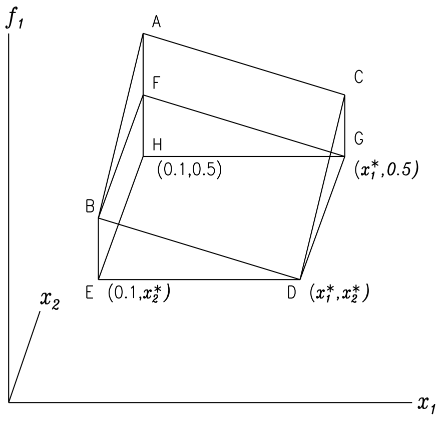


Fig. 3.8 Newton's method, estimation of derivatives of $f(x_1, x_2)$

However, if we repeat the process for f_2 and arrange for them both to have the same D, we can take this point to be the next approximation to the root.

We first consider f_1. If we keep x_2 constant in f_1 it may be regarded as a function of x_1 only and its gradient $\frac{\partial f_1}{\partial x_1}$ is given by

$$\frac{\partial f_1}{\partial x_1} = -2\sin(2x_1). \tag{3.16}$$

It follows that for $x_1 = 0.1$, $\frac{\partial f_1}{\partial x_1} = -0.39734$ and so by the same argument that we used in the Newton–Raphson method for equating the gradient with the tangent of the angle

$$-0.39734 = \frac{\text{FH}}{0.1 - x_1^*}. \tag{3.17}$$

Similarly, if we keep x_1 constant in f_1 it may be regarded as a function of x_2 only and its gradient $\frac{\partial f_1}{\partial x_2}$ is given by

$$\frac{\partial f_1}{\partial x_2} = 2\sin(2x_2). \tag{3.18}$$

It follows that for $x_2 = 0.5$, $\frac{\partial f_1}{\partial x_1} = 1.6825$ and so

$$1.6825 = \frac{\text{AF}}{0.5 - x_2^*}. \tag{3.19}$$

Since AH = AF + FH and AH = $f_1(0.1, 0.5) = 0.03976$ we have

$$0.03976 = -0.39734\bigl(0.1 - x_1^*\bigr) + 1.6825\bigl(0.5 - x_2^*\bigr). \tag{3.20}$$

Equation (3.20) is an application of the Mean Value Theorem which states that under fairly general conditions a function $f = f(x_1, x_2)$ for which (x_1^*, x_2^*) is a root may be approximated by (3.21)

$$f(x_1, x_2) = \frac{\partial f}{\partial x_1}\bigl(x_1 - x_1^*\bigr) + \frac{\partial f}{\partial x_2}\bigl(x_2 - x_2^*\bigr). \tag{3.21}$$

Table 3.10 Results using Newton's method

x_1	x_2	f_1	f_2
0.1	0.5	0.03976	0.24280
0.15259	0.48879	−0.00524	0.00108
0.15653	0.49338	−0.00001	−0.00003

The closer (x_1, x_2) to (x_1^*, x_2^*) the better the approximation. Writing $x_1^* = x_1 + \delta_1$ we have from (3.20)

$$0.03976 = 0.39734\delta_1 - 1.6825\delta_2. \tag{3.22}$$

Equation (3.22) provides one relationship between x_1^* and the increment δ_1 required to find x_2^*. To obtain another, and so determine unique values, we apply the same arguments to f_2. The partial derivatives of f_2 are given by (3.23) and (3.24) which at $x_1 = 0.1$ and $x_2 = 0.5$ evaluate to -3.9601 and 3.0806 respectively.

$$\frac{\partial f_2}{\partial x_1} = -2 - 2\cos 2x_1 \tag{3.23}$$

$$\frac{\partial f_2}{\partial x_2} = \quad 2 + 2\cos 2x_2. \tag{3.24}$$

Since $f_2(0.1, 0.5) = 0.24280$ applying a similar analysis we have

$$0.24280 = 3.9601\delta_1 - 3.0806\delta_2. \tag{3.25}$$

Equations (3.22) and (3.25) give the 2×2 system:

$$\begin{pmatrix} 0.39734 - 1.6825 \\ 3.9601 - 3.0806 \end{pmatrix} \begin{pmatrix} \delta_1 \\ \delta_2 \end{pmatrix} = \begin{pmatrix} 0.03976 \\ 0.24280 \end{pmatrix}.$$

Solving this system gives $\delta_1 = 0.05259$, $\delta_2 = -0.01121$ and so $x_1^* = 0.15259$ and $x_2^* = 0.48879$. The whole process is repeated using these values to find further approximations until the values of the functions f_1 and f_2 at some point are sufficiently small for our purposes. The sequence of approximations is shown in Table 3.10. The method we have described is Newton's method for a 2×2 nonlinear system and is summarised in Table 3.11. A worked example follows.

Problem

Find a solution to the system of equations:

$$\begin{aligned} x\cos y + y\cos x &= 0.9 \\ x\sin y + y\sin x &= 0.1. \end{aligned}$$

Solution

We write the equations as $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$, where

$$f_1(x_1, x_2) = x_1\cos x_2 + x_2\cos x_1 - 0.9$$

$$f_2(x_1, x_2) = x_1\sin x_2 + x_2\sin x_1 - 0.1.$$

Table 3.11 Newton's method for 2×2 nonlinear system

	Newton's method for the 2×2 nonlinear system $f_1(x_1, x_2) = 0$ $f_2(x_1, x_2) = 0$
1	Choose a starting point x_1, x_2
2	Calculate $f_1(x_1, x_2)$ and $f_2(x_1, x_2)$
3	Test for convergence
4	Calculate the partial derivatives of f_1 and f_2 at (x_1, x_2)
5	Solve the 2×2 system $\delta_1 \frac{\partial f_1}{\partial x_1} - \delta_2 \frac{\partial f_1}{\partial x_2} = -f_1(x_1, x_2)$ $\delta_2 \frac{\partial f_2}{\partial x_1} - \delta_2 \frac{\partial f_2}{\partial x_2} = -f_2(x_1, x_2)$ for δ_1 and δ_2
6	Set x_1 to $x_1 + \delta_1$, x_2 to $x_2 + \delta_2$
7	Repeat from step 2

The partial derivatives of f_1 and f_2 are given by

$$\frac{\partial f_1}{\partial x_1} = \cos x_2 - x_2 \sin x_1 \qquad \frac{\partial f_1}{\partial x_2} = -x_1 \sin x_2 + \cos x_1$$
$$\frac{\partial f_2}{\partial x_1} = \sin x_2 + x_2 \cos x_1 \qquad \frac{\partial f_2}{\partial x_2} = x_1 \cos x_2 + \sin x_1.$$

We take $x_1 = 0$ and $x_2 = 1$ as an initial approximation to the solution. This is a reasonable choice since $f_1(0, 1) = 0.1$ and $f_2(0, 1) = -0.1$, both values being fairly close to the root. At the point $x_1 = 0, x_2 = 1$ we have

$$\frac{\partial f_1}{\partial x_1} = 0.5403, \qquad \frac{\partial f_1}{\partial x_2} = 1, \qquad \frac{\partial f_2}{\partial x_1} = 1.8415, \qquad \frac{\partial f_2}{\partial x_2} = 0$$

and so the increment (δ_1, δ_2) to be added to the current solution is found by solving the equations

$$\begin{aligned} 0.5403\delta_1 + \delta_2 &= -0.1 \\ 1.8415\delta_1 \qquad &= \ \ 0.1. \end{aligned}$$

This gives $\delta_1 = 0.05430$, $\delta_2 = -0.12934$ and so the next estimate is $x_1 = 0.05430$, $x_2 = 0.87066$ at which point $f_1 = 0.00437$ and $f_2 = -0.01121$. Clearly we are getting closer to a solution. At the point $x_1 = 0.05430$, $x_2 = 0.87066$ we have

$$\frac{\partial f_1}{\partial x_1} = 0.59707, \qquad \frac{\partial f_1}{\partial x_2} = 0.95700, \qquad \frac{\partial f_2}{\partial x_1} = 1.6341, \qquad \frac{\partial f_2}{\partial x_2} = 0.08927$$

and so the next approximation to the solution is found by solving the equations

$$\begin{aligned} 0.59707\delta_1 + \quad 0.957\delta_2 &= -0.00437 \\ 1.6341\delta_2 + 0.08927\delta_2 &= \quad 0.1121. \end{aligned}$$

This gives the solution $\delta_1 = 0.00736$, $\delta_2 = -0.00915$ and so the next estimate is $x_1 = 0.06167$, $x_2 = 0.86151$ at which point $f_1 = 0.00030$ and $f_2 = -0.00011$. One more iteration produces a solution $x_1 = 0.06174$ and $x_2 = 0.86143$ which gives f_1 and f_2 values of order $1.0_{10} - 8$.

3.7.1 Higher Order Systems

Newton's method readily extends to larger systems, although to write out the details in the way we have done for a 2×2 system would be rather cumbersome. To simplify the detail we use vector notation.

Consider the $n \times n$ system of (nonlinear) equations

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0. \end{aligned}$$

We introduce the vectors $\mathbf{f} = (f_1, f_2, \ldots, f_n)^T$ and $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ to write the equations in the condensed form as $\mathbf{f}(\mathbf{x}) = \mathbf{0}$. As a further simplification to the notation we introduce the **Jacobian** of a function $\mathbf{f}$. The Jacobian of n functions $f_1, f_2, \ldots, f_n$, all of n variables $x_1, x_2, \ldots, x_n$ is defined to be the $n \times n$ matrix $J(\mathbf{x})$ where the (i, j)th element of J is given by $\frac{\partial f_i}{\partial x_j}$. Using this notation, and using $\boldsymbol{\delta}$ to denote the increments $\delta_1, \ldots, \delta_n$ applicable to improve current estimates, $\mathbf{x}$ to the solution, Newton's method for an $n \times n$ nonlinear system is summarised in Table 3.12. A worked example follows.

Problem

Use Newton's method to find a solution to the equations

Table 3.12 Newton's method for an $n \times n$ nonlinear system

	Newton's method for the $n \times n$ nonlinear system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$
1	Choose a starting point $\mathbf{x}$
2	Calculate $\mathbf{f}(\mathbf{x})$
3	Test for convergence
4	Calculate $J(\mathbf{x})$
5	Solve for $\mathbf{x}^*$ the $n \times n$ system $J(\mathbf{x})\boldsymbol{\delta} = -\mathbf{f}(\mathbf{x})$
6	Set $\mathbf{x} = \mathbf{x} + \boldsymbol{\delta}$
7	Repeat from step 2

$$3+\frac{2}{r^2}-\frac{1}{8}\frac{(3-2v)}{1-v}\omega^2r^2=4.5$$
$$6v-\frac{1}{2}\frac{v}{1-v}\omega^2r^2=2.5$$
$$3-\frac{2}{r^2}-\frac{1}{8}\frac{(1+2v)}{1-v}\omega^2r^2=0.5.$$

These equations arise in the modelling of stress on a turbine rotor.

Solution

Modifying the notation so that v, ω and r become x_1, x_2 and x_3 respectively, and introducing $\mathbf{f}=(f_1, f_2, f_3)^T$ we solve the system $\mathbf{f}(\mathbf{x})=\mathbf{0}$, where

$$f_1=3+\frac{2}{x_3^2}-\frac{1}{8}\frac{(3-2x_1)}{1-x_1}x_2^2x_3^2-4.5$$
$$f_2=6x_1-\frac{1}{2}\frac{x_1}{1-x_1}x_2^2x_3^2-2.5$$
$$f_3=3-\frac{2}{x_3^2}-\frac{1}{8}\frac{(1+2x_1)}{1-x_1}x_2^2x_3^2-0.5.$$

The Jacobian of these functions is given by

$$\mathbf{J}=\begin{pmatrix} -\frac{x_2^2x_3^2}{8(1-x_1)^2} & -\frac{(3-2x_1)x_2x_3^2}{4(1-x_1)} & -\frac{4}{x_3^3}-\frac{(3-2x_1)x_2^2x_3}{4(1-x_1)} \\ 6+\frac{x_2^2x_3^2}{2(1-x_1)^2} & -\frac{x_1x_2x_3^2}{1-x_1} & -\frac{x_1x_2^2x_3}{1-x_1} \\ -\frac{3x_2^2x_3^2}{8(1-x_1)^2} & -\frac{(1+2x_1)x_2x_3^2}{4(1-x_1)} & \frac{4}{x_3^3}-\frac{(1+2x_1)x_2^2x_3}{4(1-x_1)} \end{pmatrix}.$$

Starting with estimates $x_1=0.75$, $x_2=0.5$, $x_3=0.5$, we find that

$$\mathbf{J}=\begin{pmatrix} -0.1250 & -0.0469 & -16.1875 \\ 6.5000 & -0.3750 & -0.3750 \\ -0.3750 & -0.3125 & 15.6875 \end{pmatrix}.$$

To find the next estimate we use Gaussian elimination to solve the 3×3 linear system

$$\mathbf{J}(\mathbf{x})\boldsymbol{\delta}=-\mathbf{f}(\mathbf{x})$$

where $\mathbf{x}$ is the current, to find $\boldsymbol{\delta}=(-0.1497, 2.0952, 0.3937)$ which we add to $\mathbf{x}$ to give the next estimate. The process is repeated until function values f_1, f_2 and f_3 are sufficiently small. Results are shown in Table 3.13.

Table 3.13 Newtons' method for a 3 × 3 system

Step	x_1	x_2	x_3	f_1	f_2	f_3
1	0.7500	0.5000	0.5000	6.4531	1.9063	−5.5781
2	0.6003	2.5952	0.8937	−6.4531	−1.9063	5.5781
3	0.6052	1.0042	1.1290	2.0237	2.9384	3.7066
4	0.5627	0.9790	0.9942	0.6595	−0.1459	−0.0313
5	0.5305	0.9661	1.0025	−0.0159	−0.2666	0.0989
	⋮					⋮
13	0.5001	0.9999	1.0000	0.0001	−0.0007	0.0000
14	0.5000	1.0000	1.0000	0.0000	−0.0003	0.0000
15	0.5000	1.0000	1.0000	0.0000	−0.0001	0.0000

Summary In this chapter we have looked at four methods for finding a root of a general nonlinear equation $f(x) = 0$, namely bisection, false position, the secant method and the Newton–Raphson method. We have seen that the principle differences between these methods are

- the number of starting values.
  - the bisection, false position and secant methods require 2 initial root estimates.
  - the Newton–Raphson method requires 1 initial estimate only.
- the guarantee, or otherwise, of convergence.
  - the bisection and false position methods give guaranteed convergence.
  - the secant method does not give guaranteed convergence.
  - the Newton–Raphson method gives guaranteed convergence if the initial estimate is sufficiently close to a root.
- the speed of convergence.
  - the bisection method gives the slowest rate of convergence.
  - the Newton–Raphson method gives the fastest rate of convergence.
  - the convergence rate of the false position and secant methods is between that of the bisection method and the Newton–Raphson method.
- the cost per iteration.
  - the bisection, false position and secant methods involve 1 function evaluation per iteration.
  - the Newton–Raphson method involves 2 function evaluations per iteration (counting a derivative evaluation as a function evaluation). But it should be remembered that Newton–Raphson is generally faster to converge and so overall should involve fewer function evaluations.

If speed is of the essence (for example in a real time application) the bisection method may be too slow to be of practical use. Although generally faster than bisection the non-guaranteed convergence of the secant method is a serious drawback. If f can be differentiated analytically and function evaluations are not too time consuming to compute, then the Newton–Raphson method is probably the one to choose, despite the fact that even here convergence is not guaranteed unless a start is made sufficiently close to a root.

Finally we looked at Newton's method for systems of equations which we showed to be an extension of the Newton–Raphson method.

Exercises

1. Write a Matlab program to find $\sqrt{2}$ using the bisection method. Do this by finding a root of the function $f(x) = x^2 - 2$ in the interval $[1, 2]$. Aim for an accuracy of 4 decimal places. Write Matlab code based on the following which shows the interval containing the root shrinking to a point at which it has length less than 0.00005. The code assumes that the function $sq2(x)$ to evaluate $x^2 - 2$ has been established in an M-file *sq2.m* (see Sect. 1.5).

```
low = 1;  % initial lower bound
high = 2;  % initial upper bound
while high− low > 0.00005;
  mid = (high + low)/2;  % mid point of current bounds
  if sq2(low)∗sq2(mid) < 0  % solution lies to the left
     high = mid;
  else % solution lies to the right
    low = mid;
  end;
  s = sprintf('solution lies in [%8.4f,%8.4f]', low, high);
  disp(s);
end;
```

2. Use the following code to plot the function $f(x) = x \sin x - \cos x$ to show that it has just one root in the interval $[0, \pi/2]$.

```
% construct 50 x-points equally spaced between 0 and pi/2
  x = linspace(0, pi/2, 50);
% use element by element multiplication to form x sin(x)
  y = x.∗sin(x) - cos(x);
  plot(x,y);
```

   Modify the code from question 1 to find the root of $f(x) = x \sin(x) - \cos(s)$ in $[0, \pi/2]$ to an accuracy of four decimal places. Check the result using the Matlab function *fzero*. A suitable command would be

```
solution = fzero(@fsincos, 1.0)
```

   where the function $fsincos(x) = x \sin(x) - \cos(x)$ is defined in the M-file fsincos.m. Note how the name of the function, *fsincos* is passed to *fzero* using the @ symbol. The second parameter of *fzero* supplies a starting estimate of the solution.

3. Modify the code for the bisection method (question 1) to implement the method of false position. Replace the variable name *mid* by a more meaningful name such as *xstar*, where *xstar* is calculated using the (3.7). Modify the condition for the *while loop* to terminate. Test the code by finding $\sqrt{2}$ to an accuracy of 4 decimal places.

4. Use the following code as the basis for writing a Matlab program to find a root of the function $f(x) = 0$ using the secant method.

```
% Secant method for solving f(x)=0
% Assume estimates a and b, the required accuracy eps and function f have
% been defined
  while abs(f(xstar)) > eps
    xstar = b - f(b)*(a-b)/(f(a)-f(b));
    a = b;
    b = xstar;
  end;
  disp(xstar);
```

   Test the program by finding $\sqrt{2}$ to 4 decimal places using starting estimates (i) $a = 1, b = 2$ to find the positive root and (ii) $a = -1, b = -2$ to find the negative root.

5. By plotting a suitable graph show that the polynomial $4x^3 - 8x^2 + 3x - 10$ has only one real root. Use the secant program developed in question 4 to find this root. Display successive approximations to show progress to the solution. Because of the nature of the function as shown in the plot, progress may be rather slow for certain initial estimates. For estimates $a = -0.5$, $b = -1$ the method fails, but with a plot to hand this would be an unlikely choice.

   Check the solution using the Matlab function *roots* which finds all the roots of a polynomial. The input to the function is a vector of coefficients in left to right order. For the quoted example a Matlab command

   roots([4 −8 3 −10])

   would be appropriate.

6. Write a Matlab program to find $\sqrt{2}$ using the Newton–Raphson method. As before do this by finding a root of the function $f(x) = x^2 - 2$. You will see that the method is equivalent to continually replacing the current estimate x, of $\sqrt{2}$, by a new estimate $(\frac{x}{2} + \frac{1}{x})$.

   Consider the more general problem of finding the pth root of a number a, (p, a positive real numbers) by using the Newton–Raphson method to find a root of the function $f(x) = x^p - a$. Test the program on examples such as $8^{1/3}$ and $2^{2/5}$. Compare with the values expressed by Matlab commands.

7. Write a general purpose Matlab function Newton–Raphson to find a root of a given function f, where f and its derivative f' are defined as Matlab functions in the M-files f.m and fdash.m respectively. The Newton–Raphson function is to have input parameters *estimate*, an estimate of the solution, *abs* the required absolute accuracy of the solution and *limit* a limit on the number of iterations.

   The following is a suggestion for the form of the function:

```
function [root] = Newton–Raphson(estimate, eps, limit)
  x = estimate;
  count = 0;
  while diff > eps && count < limit
```

```
    if fdash(x) ~= 0
    root = x - f(x)/fdash(x);
    diff = abs(root-x);
    x = root;
    count = count + 1;
  else
 % Print a diagnostic message to show why the procedure has failed
    s = sprintf('Method fails, zero derivative at %8.4f', x);
    disp(s);
    break;
  end;
```

Notice how the *break* command has been used to return prematurely from the *while* loop and in this case return to the calling program.

Test the program using the function of question 5. Notice how Newton–Raphson generally requires fewer iterations than the Secant method.

8. Write a Matlab program to find a solution to the following system of equations. Use Newton's method for a 2 × 2 system following the scheme to verify the results shown in Sect. 3.7

$$x_1 \cos x_2 + x_2 \cos x_1 = 0.9$$
$$x_1 \sin x_2 + x_2 \sin x_1 = 0.1.$$

A suitable program might have the form

```
x = [0 ; 1]   % initial estimate of the solution
for i = 1 : 3   % perform three iterations
  % form the matrix of coefficients
  A = [diff1(x,1) diff1(x,2) ; diff2(x,1) diff2(x,2)];
  % right hand side
  b = −[ f1(x) ; f2(x)]
  d = A\b;   % solve the equations
  x = x + d   % next estimate
end;
```

Matlab provides the function *fsolve* to find a solution of a system of non-linear equations. The function provides many options but for this example the following program would be appropriate.

```
x0 = [ 0 ; 1 ];   % initial estimate
options=optimset('Display','iter');   % options to display the iterative process
[x] = fsolve(@sys,x0,options)   % solution returned in vector x
```

where it is assumed that the system of equations is defined as the function *sys* in the Matlab M-file *sys.m* as follows

```
function eqns = sys(x)
  eqns = [ x(1)*cos(x(2)) + x(2)*cos(x(1)) −0.9;
           x(1)*sin(x(2)) + x(2)*sin(x(1)) −0.1 ];
```

9. Write a Matlab program to find a solution to the following system of equations. Use Newton's method for an $n \times n$ system following the scheme shown in Table 3.12.

$$\begin{aligned} 4x_1^2 + 2x_2 &= 4 \\ x_1 + 2x_2^2 + 2x_3 &= 4 \\ x_2 + 2x_3^2 &= 4. \end{aligned}$$

Alternatively use the Matlab function *fsolve* as described in the previous question with the equations stored in column vector form as shown below.

```
% storing the equations in column vector eqns:
eqns = [ 4*x(1)^2 + 2*x(2) - 4;
         x(1) + 2*x(2)^2 + 2*x(3) - 4;
         x(2) + 2*x(3)^2 - 4 ].
```

Chapter 4
Curve Fitting

Aims In this chapter we look at ways of fitting a smooth curve to supplied data points. That is, given a set of pairs of values (x_i, y_i), we construct a continuous function $y = f(x)$ that in some sense represents an underlying function implied by the data points. Having produced such an approximating function we could for example, estimate a value for $f(x)$ where x is not one of the x_i.

Overview Initially we consider using a straight line as the approximating function. We then consider using polynomials as approximating functions and how we might find appropriate values for polynomial coefficients. Finally we consider using more general functions for approximating purposes and in so doing introduce the method of *least squares approximation*. In developing the various methods we take account of whether the approximating function, $f(x)$ is required to fit the data exactly or not. It may be possible to produce a simpler, smoother approximation and one more amenable to further analysis and application by allowing a more approximate fit to the data. In practice this is not an inappropriate approach as data collected from observation or experiment may contain error or may have limited accuracy. In producing an underlying approximation having such desirable features we may be improving the accuracy of the data and possibly identifying and eradicating error.

Acquired Skills After reading this chapter you will understand how data may be represented by a function which may then be used to approximate in-between and beyond the given data. You will be able to assess the type of function required, whether linear, polynomial or other, in a given situation. You will be able to construct the function, choosing from one of a number of methods appropriate to the problem under consideration.

4.1 Introduction

Curve fitting is the process whereby a continuous function is constructed in such a way that it represents supplied data. The main problem is that of deciding what form the continuous function should take. A linear function may be suitable for the measurement of temperature, but not for measuring gas volume against pressure. For the latter we find that a polynomial is a more suitable function. It may be tempting to assume that in general a higher order polynomial is likely to offer a better approximation than a polynomial of lower order, but this is not necessarily so. Indeed, the converse may apply. However, rather than discard polynomials entirely, we show how they can be combined in a piecewise manner to form good approximating functions. Such functions are known as spline functions. We begin by using straight line (**linear approximation**) to fit exactly to given data.

4.2 Linear Interpolation

The temperature of a liquid can be determined by reading a non-digital mercury thermometer. The length of the mercury column may well coincide with one of the calibration points, but more typically it will lie somewhere between two such points (a lower and an upper bound).

Problem

How might we measure temperatures using a non-digital mercury thermometer which is marked at 10° intervals in the range 0° to 100°?

Solution

We estimate the length of the mercury column above the lower point as a fraction of the distance between the lower point and the upper point. The temperature of the liquid is assumed to be the value at the lower of the two points plus the fraction times the interval between two calibrations (10°).

A graphical representation of this process (Fig. 4.1) is an x-y plot in which the y-axis represents the length of the mercury column and the x-axis represents temperature. At the calibration points (temperatures in the range 0° to 100° in steps of 10°) we plot the known values of the length of the mercury column on a graph. Assuming suitable consistency conditions are satisfied, such as the cross-section of the column is constant throughout the scale portion of the thermometer, the relationship should be linear and we should therefore be able to draw a straight line that (to within the limits of experimental accuracy) links all of the points. Then for any known length of the mercury column we simply draw a horizontal line across from the appropriate point on the y-axis, and where this meets our straight line we draw a vertical line; the point at which this line crosses the x-axis defines the temperature. In this case the fitted curve (albeit a straight line) is a representation of the data supplied as lengths of the mercury column (the function values) at temperatures (the independent variable) in the range 0° to 100° in steps of 10°.

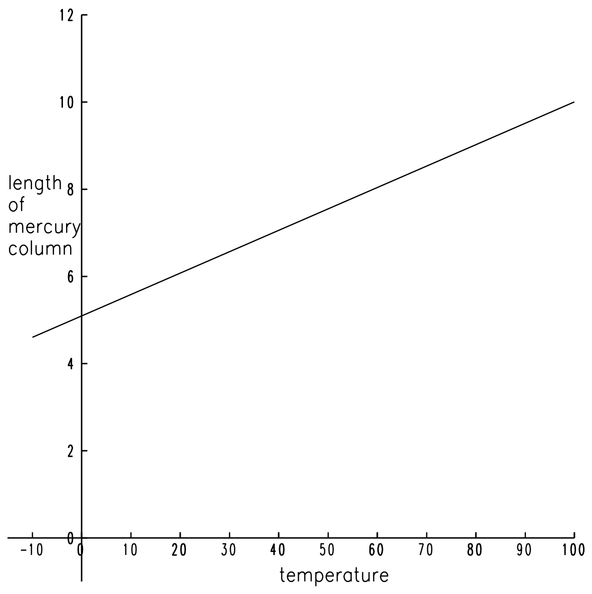


Fig. 4.1 Length of mercury column against temperature

Discussion

There are a number of interesting points to note. Since we are dealing with a straight line it is possible to read off values on the y-axis which correspond to values on the x-axis. Similarly it is possible to read off values on the x-axis which correspond to values on the y-axis. This process known is as **inverse interpolation**.

It is perfectly possible to extend the straight line and perform interpolation and inverse interpolation for values of the temperature and length of the mercury column that lie outside of the range of values supplied as data. The process is known as **extrapolation**, which can be a risky proposition. Consider what would happen if the independent variable were a time variable. Whereas drawing information from times past might be a fairly reliable activity, predicting the future is a very different matter as many have found to their cost.

The measurement of temperature using our hypothetical thermometer is an example of linear interpolation, in that the curve fitted to the data is a straight line. We now consider another problem (which we will subsequently refer to as the *gas* problem) and discuss the question of deciding the suitability, or otherwise, of linear interpolation for a given set of data.

Problem

From a set of experiments the values in Table 4.1 have been obtained relating the volume of a gas, y to pressure, x. The aim is to determine from this data volumes of the gas at non-tabulated pressure results and an estimate of the underlying function of the form shown in Fig. 4.2.

Table 4.1 Volume of gas in relation to pressure

y	1.62	1.00	0.75	0.62	0.52	0.46
x	0.5	1.0	1.5	2.0	2.5	3.0

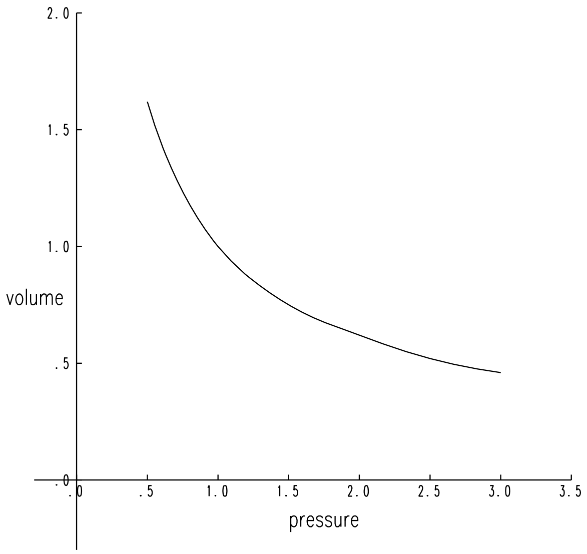


Fig. 4.2 Graph of gas volume against pressure

Solution

As with the example of the thermometer, we assume that between any two tabulated points the volume versus pressure graph behaves linearly. Thus, for example, if we are interested in determining the volume at pressure $x = 1.75$, then since 1.75 is mid-way between the tabulation points 1.5 and 2.0, then the volume is, approximately, mid-way between the observations 0.75 and 0.62, and we deduce a value of 0.685. In general, to compute a volume $\hat{y}$ corresponding to the pressure $\hat{x}$, we find the i such that $x_i \leq \hat{x} \leq x_{i+1}$, where $x_0 = 0.5$, $x_1 = 1.0, \dots, x_5 = 3.0$. If we use y_i to denote the volume corresponding to pressure x_i, then our estimate of the volume $\hat{y}$ corresponding to pressure $\hat{x}$ is given by

$$\hat{y} = y_i + \frac{\hat{x} - x_i}{x_{i+1} - x_i}(y_{i+1} - y_i). \tag{4.1}$$

Discussion

Strictly speaking, we have described **piecewise linear interpolation**. Rather than compute a single straight-line approximation that goes through all the data points we produce a sequence of approximations, with each approximation passing through two consecutive data points. We are not interested in any of these approximations outside of its range. For an appropriate problem it should be possible, in theory, to obtain the form of a single straight line that passes through all the data points (our thermometer problem can be classified as an appropriate problem). In practice the situation is less clear; there may well be observational error in the data.

4.2.1 Differences

The accuracy of piecewise linear interpolation will depend on the accuracy of the supplied data and that the relationship between dependent and independent variables is linear. The validity or otherwise of the latter can be deduced by forming what are known as divided differences (or just **differences**), and this in turn provides the route to alternative methods of solution. We illustrate the method by constructing a table of differences for the following example.

In an experiment designed to demonstrate the relationship between time and the average number of goals scored by a football club per game, a set of results over a playing season was obtained and summarised in Table 4.2. We might deduce that the average number of goals scored after 45 minutes is $0.8 + \frac{45-40}{50-40}(1.0 - 0.8) = 0.9$, provided that the relationship between time and goals scored is linear but that is yet to be established. Let the function values (average number of goals) be y_i corresponding to the points (time duration) x_i. Then the divided difference involving the two consecutive values y_{i+1} and y_i is simply the difference $y_{i+1} - y_i$ divided by $x_{i+1} - x_i$. Divided differences corresponding to the data of Table 4.2 are shown in the third column of Table 4.3; we observe that they all have the same value (0.02). The divided differences themselves can be regarded as y-values, and hence we can construct another set of differences (second differences), shown in the fourth column of Table 4.3. Note that in order to produce these second differences we must fan back to the first column to determine the divisor. Hence, for example, the first entry in the final column is found as $\frac{0.2-0.2}{20-0}$. Since the first differences are identical, the second differences are all zero (and so here the actual form of the divisor is not important).

To realise the importance of the difference table, we know from Taylor's Theorem[1] that the first derivative of a function can be approximated by the difference between two function values divided by the difference between values of the points at which those function values were obtained. Obtaining a set of values in this way, it follows that second derivative approximations can be obtained by forming divided differences of first divided differences. If these second differences are

Table 4.2 Average number of goals scored per game

Goals	0.0	0.20	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8
Play in minutes	0	10	20	30	40	50	60	70	80	90

[1]Brook Taylor 1685–1731, published his theorem in 1715 although others were able to make similar claims. The theorem began to carry his name some fifty years later. He somewhat aggressively promoted Newton's work over that of Leibniz and other European mathematicians.

Table 4.3 Differences for goals against time

Duration of play	Av. no. goals	1st differences	2nd differences
0	0.00		
		0.02	
10	0.2		0.0
		0.02	
20	0.4		0.0
		0.02	
30	0.6		0.0
		0.02	
40	0.8		0.0
		0.02	
50	1.0		0.0
		0.02	
60	1.2		0.0
		0.02	
70	1.4		0.0
		0.02	
80	1.6		0.0
		0.02	
90	1.8		

Table 4.4 Differences for volume against pressure

Pressure	Volume	Differences				
		1st	2nd	3rd	4th	5th
0.5	1.62					
		−1.24				
1.0	1.00		0.74			
		−0.50		−0.333333		
1.5	0.75		0.24		0.106667	
		−0.26		−1.20		−0.016
2.0	0.62		0.06		0.066667	
		−0.20		0.133333		
2.5	0.52		0.08			
		−0.12				
3.0	0.46					

zero (and it follows that all further differences must also be zero), then the suggestion is that the data has been sampled from a function having zero second derivatives (that is, a straight line) and so is best represented by a straight line. However the validity of such a conclusion will depend on the frequency of the sampling points.

If we consider the data relating to the *gas* problem (Table 4.1) we can form first, second and other divided differences to form the results shown in Table 4.4. From the comments we made concerning Table 4.3 we conclude that a single straight line cannot be found which exactly fits all the data of our *gas* problem. However in constructing Table 4.4 we have computed additional information which tells us something about the curvature of the underlying function represented by the data values, which we are able to exploit.

4.3 Polynomial Interpolation

In the previous section we saw that straight line approximation is not sufficient in all cases. In general unless we have good reasons to do otherwise, we let the approximating function take the form of a polynomial in the independent variable. The aim therefore is to choose the polynomial coefficients appropriately.

As a first step we consider the case of **polynomial interpolation**. If the supplied data is given in the form of $n+1$ values y_i, $i = 0, 1, \ldots, n$, corresponding to values x_i, $i = 0, 1, \ldots, n$, we choose an approximating polynomial $p_n(x)$ of degree n so that

$$p_n(x_i) = y_i, \quad i = 0, 1, \ldots, n. \tag{4.2}$$

$p_n(x)$ has $n+1$ (as yet) unknown coefficients. Since in (4.2) we have the same number of equations as unknowns, in principle we may solve the system using the techniques outlined in Chap. 1. Assuming the x_i are unique, then it can be shown that the solution itself is unique. The coefficient matrix is non-singular. In practice a more indirect approach is usually taken, and we see that it is sometimes the case that an explicit expression for $p_n(x)$ is not derived.

4.3.1 Newton Interpolation

We remarked in Sect. 4.2 that a simple way of estimating non-tabulated values is to employ linear interpolation, as represented by formula (4.1). In Table 4.4 we evaluate the factors $(y_{i+1} - y_i)/(x_{i+1} - x_i)$ which appear in this formula. In order to present a general form for linear interpolation we introduce the notation

$$y[x_i, x_{i+1}] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \tag{4.3}$$

so that (4.1) becomes $\hat{y} = y_i + (\hat{x} - x_i)y[x_i, x_{i+1}]$. In effect, we have the straight line

$$p_1(x) = y_i + (x - x_i)y[x_i, x_{i+1}] \tag{4.4}$$

that passes through the pair of coordinates (x_i, y_i) and (x_{i+1}, y_{i+1}), and that evaluating this expression (which may be regarded as a polynomial of degree 1) at some point $\hat{x}$ lying between x_i and x_{i+1} gives an approximation to the corresponding value of y.

Extending the notation of (4.3) we write

$$y[x_i, x_{i+1}, x_{i+2}] = \frac{y[x_{i+1}, x_{i+2}] - y[x_i, x_{i+1}]}{x_{i+2} - x_i}. \tag{4.5}$$

For the *gas* problem (with $i = 0, 1, 2, 3$) (4.5) produces the values in the column of second divided differences in Table 4.4. It is not difficult to show that the quadratic polynomial

$$p_2(x) = y_i + (x - x_i)y[x_i, x_{i+1}] + (x - x_i)(x - x_{i+1})y[x_i, x_{i+1}, x_{i+2}] \qquad (4.6)$$

passes through each of the coordinates (x_i, y_i), (x_{i+1}, y_{i+1}) and (x_{i+2}, y_{i+2}). That it passes through the first two of these points follows immediately from our previous considerations (the first two terms are just the right-hand side of (4.4) and the final term on the right-hand side of (4.6) vanishes at x_i and x_{i+1}). The verification of interpolation at the third point requires some algebraic manipulation.

We can extend the argument further, and form the kth divided differences as

$$y[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{y[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - y[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \qquad (4.7)$$

and it can then be established that the polynomial (the **Newton interpolating polynomial**)

$$\begin{aligned} p_k(x) = y_i & \\ &+ (x - x_i)y[x_i, x_{i+1}] \\ &+ (x - x_i)(x - x_{i+1})y[x_i, x_{i+1,} x_{i+2}] \\ &+ \cdots \\ &+ (x - x_i)(x - x_{i+1})\cdots(x - x_{i+k-1})y[x_i, x_{i+1}, \ldots, x_{i+k}] \qquad (4.8) \end{aligned}$$

passes through each of the coordinates (x_i, y_i), (x_{i+1}, y_{i+1}), $\ldots$, (x_k, y_k). Evaluating this polynomial at some point $\hat{x}$ between x_i and x_{i+k} then gives an approximation to $y(\hat{x})$.

Problem

Find the polynomial (the Newton interpolating polynomial) which interpolates the data given in Table 4.1 (the *gas* problem). Use this polynomial to estimate the volume of gas at pressure 1.75.

Solution

We have 6 pairs of data (x_i, y_i), $i = 0, 1, \ldots, 5$. Using (4.8) and the appropriate values from Table 4.4 we may write the interpolating polynomial as

$$\begin{aligned} p_5(x) = 1.62 & \\ &+ (x - 0.5)(-1.24) \\ &+ (x - 0.5)(x - 1.0)(0.74) \\ &+ (x - 0.5)(x - 1.0)(x - 1.5)(-0.333333) \\ &+ (x - 0.5)(x - 1.0)(x - 1.5)(x - 2.0)(0.106667) \\ &+ (x - 0.5)(x - 1.0)(x - 1.5)(x - 2.0)(x - 2.5)(-0.016). \end{aligned}$$

It can be verified that this polynomial interpolates the data. Evaluating $p_5(1.75)$ gives 0.678672, the estimated volume at pressure 1.75.

In general if we have $n + 1$ coordinates, divided differences may be used to determine the coefficients in a polynomial of degree n, which passes through all of the points. Given the uniqueness property this polynomial must be identical to the one that we would have obtained using the direct approach of Sect. 4.3 although computing round-off error may produce discrepancies. It may be noticed that it is easy to extend the approximating polynomial if more data points are added to the original set as we will see in the following problem.

Problem

Suppose, for example, in our *gas* problem that the volume $y = 0.42$ of a gas at pressure $x = 3.5$ is made available in addition to the data shown in Table 4.1. Use this information to estimate y at $x = 1.75$.

Solution

We can immediately compute a diagonal of differences to be added to the bottom of Table 4.4 to obtain the values shown in Table 4.5. The addition of a data point means that we now have differences up to order 6, and our previous approximation to $y(1.75)$ is modified by the addition of the term

$$1.25 \times 0.75 \times 0.25 \times (-0.25) \times (-0.75) \times (-1.25) \times (-0.006222) = 0.000342$$

to give a potentially more accurate approximation of $y(1.75) = 0.678965$. We cannot categorically say that this is a more accurate approximation as there may be

Table 4.5 Additional differences for the *gas* problem

Pressure	Volume	Differences				
		1st	2nd	···	5th	6th
0.5	1.62					
		−1.24				
1.0	1.00		0.74			
		−0.50				
1.5	0.75		0.24	···		
		−0.26			−0.016	
2.0	0.62		0.06			−0.006222
		−0.20			−0.03467	
2.5	0.52		0.08	···		
		−0.12				
3.0	0.46		0.04			
		−0.08				
3.5	0.42					

local behaviour around the pressure at point 1.75 which is not reflected in the data.

4.3.2 *Neville Interpolation*

In the previous section we observed that it is possible to construct a polynomial, $p_n(x)$, of degree n which passes through a sequence of $n+1$ coordinates. Having obtained an explicit form for the polynomial, an interpolated value can then be obtained at any required point, and this process can clearly be applied over and over again.

Neville[2] **interpolation** is an alternative approach that does not yield an explicit form for the interpolating polynomial. Neville's table yields a single interpolated value, and the table has to be completely reconstructed if interpolation at some other point is required. The method is illustrated in the following problem.

Problem

Use Neville interpolation on the *gas* problem to estimate a value for $y(1.75)$ from the data supplied in Table 4.1.

Solution

For Neville interpolation a table of the form shown in Table 4.6 is constructed. Consider the following rearrangement of the equation for linear interpolation (4.1)

$$\hat{y} = \frac{(\hat{x} - x_i)y_{i+1} - (\hat{x} - x_{i+1})y_i}{x_{i+1} - x_i}, \quad i = 0, 1, \ldots, 5. \tag{4.9}$$

This formula defines the way in which the values shown in the third column of Table 4.6 (labelled *Linear*) have been calculated. Setting $\hat{x} = 1.75$ we have for $i = 0$, $((1.75 - 0.5) \times 1.00 - (1.75 - 1.0) \times 1.62)/(1.0 - 0.5) = 0.07$. Similarly, setting $i = 1$ we find a value 0.625, and so on up to values corresponding to $i = 5$.

Each entry in the *Linear* column is an estimate of $y(1.75)$. Only the third is an interpolated approximation since linear interpolation has been performed using points which bracket $\hat{x}$. All remaining estimates are based on extrapolated values.

To complete the table we repeat the process using values in one column to proceed to the next. The fourth column of values (labelled *Quadratic*) is obtained from the third column (and the x_i), the fifth column (labelled *Cubic*) from the fourth column, and so on. The x_{i+1} in (4.9) is taken to be the x value on the same row as the element to be computed. The appropriate x_i value is found by tracking back

[2]E.H. Neville, 1889–1961, among many other achievements instrumental in bringing wonder mathematician Ramanujan to Cambridge to study with the eminent pure mathematician G.H. Hardy.

Table 4.6 Interpolated values for Neville's algorithm

Pressure	Volume	Linear	Quadratic	Cubic	Quartic	Quintic
0.5	1.62					
1.0	1.00	0.070				
1.5	0.75	0.625	0.76375			
2.0	0.62	0.685	0.67000	0.685625		
2.5	0.52	0.670	0.68125	0.675625	0.679375	
3.0	0.46	0.610	0.68500	0.681875	0.677969	0.678672

Table 4.7 Additional row for Neville's table

x	y	1	2	3	4	5	6
3.5	0.43	0.535	0.66625	0.688125	0.682656	0.679375	0.678965

diagonally from the position of the value to be computed to the function value in the second column. The corresponding x-value in the first column is the one that is required.

For example, the value in row 6, column 5 (0.681875) is calculated using the values in column 4, rows 5 and 6 (0.68125 and 0.68500), and the values in column 1 in rows 3 and 6 (1.5 and 3.0) to give $((1.75 - 1.5) \times 0.68500 - (1.75 - 3.0) \times 0.68125)/(3.0 - 1.5)$.

The value in the final column (labelled *Quintic*) gives the interpolated value based on all of the data. Therefore by Neville interpolation we have the estimate $y(1.75) = 0.678672$, the same value obtained using Newton's method.

Discussion

Mathematically speaking, the uniqueness condition says that the value we obtain by this approach is identical to the one we would have obtained by calculating the polynomial coefficients via the solution of a system of equations or Newton's method, and then evaluating either of those polynomials at $x = 1.75$ although rounding error in computing may introduce discrepancies.

As with Newton interpolation, the table associated with Neville interpolation can be readily extended by the addition of one or more data points. For example, if, as before, we include the additional point $(3.5, 0.43)$, then the effect is to add a row to the bottom of Table 4.6 with contents given in Table 4.7.

4.3.3 A Comparison of Newton and Neville Interpolation

To simplify the discussion we introduce a labelling for the entries in the Neville table of differences. The entries for $n = 5$ in relation to finding an approximation for $\hat{y} = y(\hat{x})$, $\hat{x} = 1.75$ are shown in Table 4.8.

Table 4.8 Labelling of interpolated values for Neville's algorithm

x	y	Linear	Quadratic	Cubic	Quartic	Quintic
x_0	$y_0 = P_{00}$					
x_1	$y_1 = P_{10}$	P_{11}				
x_2	$y_2 = P_{20}$	P_{21}	P_{22}			
x_3	$y_3 = P_{30}$	P_{31}	P_{32}	P_{33}		
x_4	$y_4 = P_{40}$	P_{41}	P_{42}	P_{43}	P_{44}	
x_5	$y_5 = P_{50}$	P_{51}	P_{52}	P_{53}	P_{54}	P_{55}

Neville's algorithm, as we have presented it, could be adapted to yield an explicit form for an approximating polynomial, but the entries in the difference table need to be polynomials, rather than just numbers. For example, the entries in the first column would be linear polynomials, rather than the values of those polynomials evaluated at an interpolation point, as given in Table 4.6. From (4.9) we deduce that the first entry in this column would be

$$P_{11}(x) = \frac{(x - x_0)y_1 - (x - x_1)y_0}{x_1 - x_0}$$

whilst the second entry $(P_{21}(x))$ would be

$$P_{21}(x) = \frac{(x - x_1)y_2 - (x - x_2)y_1}{x_2 - x_1}.$$

Looking to the second column, it follows that the first entry would be computed as

$$P_{22}(x) = \frac{(x - x_0)P_{21}(x) - (x - x_2)P_{11}(x)}{x_2 - x_0} \tag{4.10}$$

and it is clear that this must be a quadratic polynomial. Further, substituting x_i, $i = 1, 2, 3$ for x in (4.10) it can be seen that $P_{22}(x_i) = y_i$. When evaluated at the point $\hat{x}$, this polynomial will give the value $P_{22} = 0.76375$ of Table 4.6.

It is easy to verify that all of the entries in the column headed 'Quadratic' in Table 4.6 are based on quadratic interpolation, in the same way that $P_{22}(\hat{x}) = P_{22}$. Similarly, it can be easily shown that entries in subsequent columns are based on cubic, quartic and quintic interpolation. It is not surprising therefore, that the potential most accurate result produced by Neville interpolation is identical to that obtained using Newton interpolation.

Mathematically there is nothing to choose between Neville and Newton interpolation, but the two methods have different computational characteristics. If a single interpolated value only is required then Neville interpolation will involve considerably fewer arithmetic operations than forming and then evaluating the interpolation polynomial using Newton's divided difference method. If several interpolated values are required then forming the interpolation polynomial and evaluating it is likely to be more effective than repeatedly constructing in full the table of Neville's algorithm. Further, the form of the polynomial provided by Newton's method is cheaper to evaluate than that provided by Neville interpolation.

4.3.4 Spline Interpolation

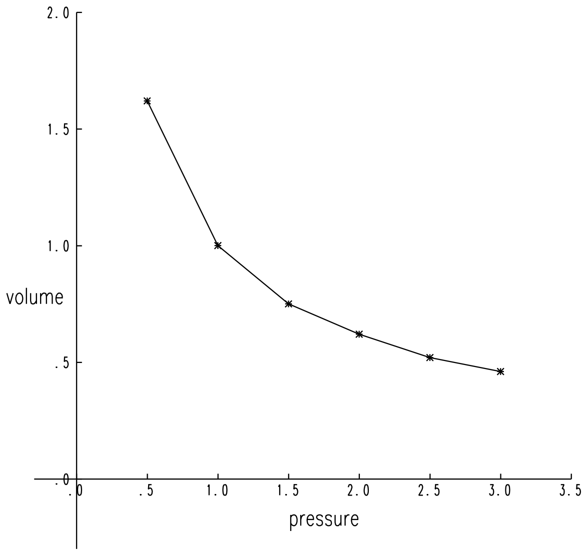
Interpolating a set of points by a polynomial is, intuitively, an attractive proposition. We would hope that as the number of points increases and, as a consequence, the order of the polynomial increases, the accuracy of an interpolated value obtained will also increase. Unfortunately there is no guarantee of this. Further, the computational cost of constructing and evaluating high-order polynomial approximations is not inconsiderable. Even if the polynomial is not formed explicitly, as in Neville interpolation, the number of arithmetic operations involved in determining an interpolated value is still high. Hence there are good mathematical and computational reasons for having a low order of the approximating polynomial.

Even by eye, it is clear that it is not possible to fit a single straight line through all of the data of our *gas* problem, which was verified mathematically by constructing a divided difference table (Table 4.4) and observing that all second divided differences are non-zero. However, in Sect. 4.3.2 we observed that it is possible to construct a table of polynomials in which all the $P_{i0}(x)$ are linear interpolants, the suffix i indicating the pair of points that the straight line passes through. For our *gas* problem this situation is illustrated in Fig. 4.3. The behaviour of P_{i0} outside the range x_i to x_{i+1} is unimportant, we are only concerned with approximating in this interval. Using these linear interpolants we can define a piecewise linear approximation $p_1(x)$ as

$$p_1(x) = P_{i0}(x), \quad \text{where } x \text{ lies between } x_i \text{ and } x_{i+1}.$$

Since each P_{i0} passes through the coordinates that define its range of interest, we have a guarantee that $p_1(x)$ is continuous throughout the range 0.5 to 3.0. However,

Fig. 4.3 Piecewise linear approximation



there is a change in the value of the slope of $p_1(x)$ at each of the internal interpolation points (the points 1.0, 1.5, 2.0 and 2.5), as indicated by the different values in the column of second differences in Table 4.4.

Piecewise linear interpolation is of limited use as it yields only a crude approximation to the data. Consideration of the Neville table, with polynomials being produced rather than interpolated values, suggests a simple way of increasing the order of polynomial approximation. $P_{20}(x)$ will be a polynomial interpolating the data points (0.5, 1.62), (1.0, 1.00) and (1.5, 0.75), $P_{21}(x)$ interpolates at the points (1.0, 1.00), (1.5, 0.75) and (2.0, 0.62), and so on. We therefore have a sequence of quadratics and can define a piecewise quadratic polynomial $p_2(x)$ as being defined by one of the $P_{2i}(x)$, depending on within which of the intervals x lies. We have a slight problem here, in that for all but the first and last intervals there is a choice between two quadratics (and the form of these quadratics will be different). This dilemma can be avoided by taking the first, third, fifth, etc., but for our *gas* problem this simply replaces one problem with another. $P_{22}(x)$ and $P_{42}(x)$ can be used to define $p_2(x)$ within 0.5 to 1.5 and 1.5 to 2.5 respectively, but what about 2.5 to 3.0? Further, whilst we have upped the degree of the approximating polynomial by one (from linear to quadratic), the discontinuity of the first (and, now, second) derivative remains, so it is not as smooth as maybe we would like.

An alternative approach which increases the degree of polynomial approximation and also improves the continuity characteristics is based on the use of splines. Piecewise linear interpolation is, in fact, a simple example of spline[3] approximation. Within each interval we have a local approximating polynomial and the overall approximation is defined as the aggregate of these individual approximations. For each local polynomial of degree n, $n+1$ polynomial coefficients will need to be defined. Suppose that we choose $n = 3$ (cubic spline interpolation). We look for a piecewise cubic approximation $p_3(x)$ such that

$$p_3(x) = p_{3,i}(x), \quad \text{if } x \text{ lies between } x_i \text{ and } x_{i+1}$$

where $p_{3,i}(x)$ is an approximating cubic between x_i and x_{i+1}.

Problem

Use spline interpolation on the *gas* problem. In particular estimate $y(1.75)$.

Solution

In the first interval we wish to determine values for the coefficients a_0, b_0, c_0 and d_0 in a cubic polynomial $p_{3,0}(x) \equiv a_0 + b_0x + c_0x^2 + d_0x^3$ such that $p_{3,0}(x)$ is the approximating polynomial between $x = 0.5$ and $x = 1.0$. For $p_{3,0}(x)$ to interpolate the data, we require $p_{3,0}(0.5) = 1.62$ and $p_{3,1}(1.0) = 1.00$. This leaves two further equations to be derived so that $p_{3,0}(x)$ is uniquely determined. Similar arguments apply to $p_{3,1}(x) \equiv a_1 + b_1x + c_1x^2 + d_1x^3$, the form of $p_3(x)$ between 1.0 and 1.5,

[3] Spline: literally, a flexible strip of wood, metal or rubber.

Table 4.9 Cubic spline coefficients

i	Interval	a_i	b_i	c_i	d_i
0	$[0.5, 1.0]$	2.24	−0.88	−1.09	0.73
1	$[1.0, 1.5]$	3.64	−5.09	3.12	−0.68
2	$[1.5, 2.0]$	1.43	−0.67	0.17	−0.02
3	$[2.0, 2.5]$	0.91	0.12	−0.22	0.04
4	$[2.5, 3.0]$	2.73	−2.07	0.65	−0.07

and so on. In total we have 5 separate cubics, giving a total of $5 \times 4 = 20$ unknowns, with $5 \times 2 = 10$ equations arising from interpolation conditions, leaving $20 - 10 = 10$ still to find. Interpolation ensures continuity of $p_3(x)$, but says nothing about the derivatives of $p_3(x)$. Hence we derive further equations by forcing continuity of the first two derivatives of $p_3(x)$ at each of the internal points. For example, to ensure continuity of the first two derivatives at $x = 1.0$, we impose the conditions

$$\begin{aligned} b_0 + 2c_0(1.0) + 3d_0(1.0)^2 &= b_1 + 2c_1(1.0) + 3d_1(1.0)^2 \\ 2c_0 + 6d_0(1.0) &= 2c_1 + 6d_1(1.0). \end{aligned}$$

It follows that we have two extra conditions at each of the four internal points, a total of $4 \times 2 = 8$ equations, leaving us still 2 short. The specification of the remaining two conditions is a little arbitrary; a common choice is that the second derivative of the first cubic is zero at $x = 0.5$, and the second derivative of the last cubic is zero at $x = 3.0$. We now have a set of 20 equations in 20 unknowns to be solved for the polynomial coefficients. Interpolation is achieved by evaluation of the appropriate cubic at the specified point.

Cubic spline interpolation used on the *gas* problem with zero second derivatives imposed at the two end points produces the polynomial coefficients shown in Table 4.9. Evaluating the third of these polynomials at the point $x = 1.75$ gives $y(1.75) = 0.6812$. This value is comparable with that obtained by Newton's method, which was confirmed using Neville.

Discussion

We cannot say that cubic spline interpolation gives a better approximation than any other of the methods considered here. A good deal depends on the accuracy of the data and the underlying function to which we are trying to match our approximating straight lines and polynomials. Nevertheless spline interpolation has been used successfully in a wide variety of application areas[4].

[4]The use of splines began with the actuarial profession in the late 19th Century. What were then described as osculating (kissing) curves were more prosaically named spline functions when numerical analysts became involved in the 1940's.

4.4 Least Squares Approximation

The final form of approximation we consider has little in common with what has gone before. The distinguishing feature here is that the straight line, or polynomial, or whatever other form we choose, may not pass through any of the data points, let alone all of them. The use of such an approximation is particularly appropriate when it is known that there may be significant errors in the data; to force a curve exactly through the data when it is known that the data is inexact clearly does not make much sense. Further, we can keep the order of the polynomial low even when the number of data points is high, which as we observed in the previous section was desirable.

4.4.1 Least Squares Straight Line Approximation

Returning yet again to the *gas* problem, if we try to construct a straight line $p_1 = a + bx$ which exactly fits all the data we have the following system of equations

$$\begin{aligned} a + b(0.5) &= 1.62 \\ a + b(1.0) &= 1.00 \\ a + b(1.5) &= 0.75 \\ a + b(2.0) &= 0.62 \\ a + b(2.5) &= 0.52 \\ a + b(3.0) &= 0.46. \end{aligned}$$

This is an **overdetermined** system of equations in that we have more equations than unknowns. Since we cannot satisfy all of the equations at once we compromise and look for a solution in which all of the equations are satisfied approximately. One way of ensuring this is to minimise the sum of the squares of the differences (or **residuals**) between supplied values of the dependent variable (the y-values) and the values predicted by the approximation. Although other definitions of what constitutes a good fit are possible, the least squares approximation has some useful mathematical properties which makes it relatively easy to compute.

Problem

Fit a least squares straight line to the data of the *gas* problem. In particular estimate $y(1.75)$.

Solution

We find values of a and b such that

$$\begin{aligned} S(a,b) = &\quad (a+b(0.5)-1.62)^2 \\ &+ (a+b(1.0)-1.00)^2 \\ &+ (a+b(1.5)-0.75)^2 \\ &+ (a+b(2.0)-0.62)^2 \\ &+ (a+b(2.5)-0.52)^2 \\ &+ (a+b(3.0)-0.46)^2 \end{aligned}$$

is made as small as possible.

Partial derivatives of S with respect to a and b must be zero for a local minimum and so we have

$$\begin{aligned} \partial S(a,b)/\partial a = &\quad 2(a+b(0.5)-1.62) \\ &+ 2(a+b(1.0)-1.00) \\ &+ 2(a+b(1.5)-0.75) \\ &+ 2(a+b(2.0)-0.62) \\ &+ 2(a+b(2.5)-0.52) \\ &+ 2(a+b(3.0)-0.46) = 0 \\ \text{and} \quad \partial S(a,b)/\partial b = &\quad 2(0.5)(a+b(0.5)-1.62) \\ &+ 2(1.0)(a+b(1.0)-1.00) \\ &+ 2(1.5)(a+b(1.5)-0.75) \\ &+ 2(2.0)(a+b(2.0)-0.62) \\ &+ 2(2.5)(a+b(2.5)-0.52) \\ &+ 2(3.0)(a+b(3.0)-0.46) = 0. \end{aligned}$$

From the two equations we have

$$6a + \ 10.5b = 4.97 \tag{4.11}$$

$$10.5a + 22.75b = 6.855. \tag{4.12}$$

This set of two equations in two unknowns has a solution $a = 1.565333$ and $b = -0.421143$, and so the approximating straight line is $y = 1.565333 - 0.421143x$ (see Fig. 4.4).

Discussion

Equations (4.11) and (4.12) may be written in the form

$$an + \ bS_x = S_y \tag{4.13}$$

$$aS_x + bS_{xx} = S_{xy} \tag{4.14}$$

Table 4.10 Gas problem, linear least squares values and residuals

y	1.62	1.00	0.75	0.62	0.52	0.46
x	0.5	1.0	1.5	2.0	2.5	3.0
least squares values	1.3548	1.1442	0.9336	0.7230	0.5125	0.3019
residuals	0.2652	−0.1442	−0.1836	−0.1030	0.0075	0.1581

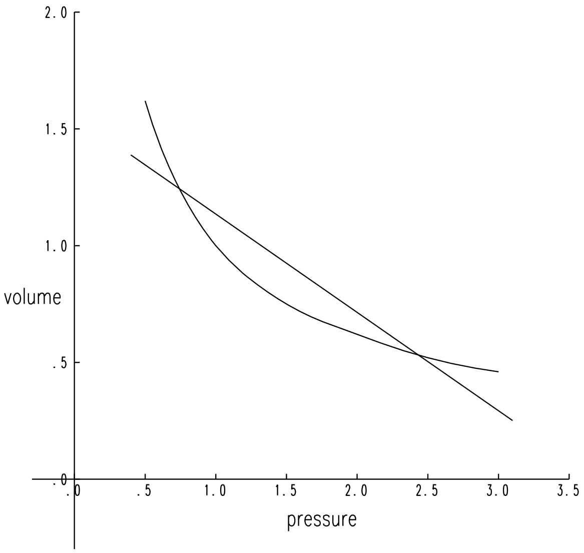


Fig. 4.4 Least squares linear approximation

where $S_x = \sum_{i=0}^{n} x_i$, $S_y = \sum_{i=0}^{n} y_i$, $S_{xx} = \sum_{i=0}^{n} x_i^2$ and $S_{xy} = \sum_{i=0}^{n} x_i y_i$ and for this example $n = 5$. These equations are known as the **normal equations**. They provide a ready means of finding the least squares straight line approximation $y = a + bx$ to n data points (x_i, y_i), $i = 0, 1, \dots, n$.

If the straight line is evaluated at each of the points x_i and the result subtracted from the corresponding y_i then we have values that indicate the amount by which the straight line fails to interpolate the given data. The results are given in Table 4.10. It can be seen that none of these values (residuals) is zero, and hence the approximating straight line does not actually pass through any of the given data points. Substituting $x = 1.75$ into our solution we obtain the approximation $y(1.5) = 0.8283$, which is out of line with Newton's and Neville's estimates of 0.6787 and the cubic spline estimate 0.6812. Moreover the sum of the squares of the residuals is 0.16053 which is not sufficiently close to zero to suggest that the data can be accurately represented by a straight line. Figure 4.4, which is a representation of the best we can do using a least-squares straight line fit is further confirmation.

4.4.2 Least Squares Polynomial Approximation

To generalise our discussion of least squares approximation we consider the polynomial approximation $p_m(x)$, of degree m, which we wish to fit to the $n+1$ data points (x_i, y_i). We assume that n is larger than m.

We cannot hope to satisfy (4.15) exactly, so as with the straight line approximation we aim to minimise the sum of squares of residuals.

$$p_m(x_i) = y_i, \quad i = 0, 1, \ldots, n \tag{4.15}$$

where

$$p_m(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_m x^m. \tag{4.16}$$

We choose the α_i so that $S(\boldsymbol{\alpha}) = \sum_{i=0}^{n} r_i^2$, where $r_i = p_m(x_i) - y_i$, is a minimum, which means we must have

$$\frac{\partial S(\boldsymbol{\alpha})}{\partial \alpha_i} = 0, \quad i = 0, 1, \ldots, m.$$

Problem

Fit a least squares quadratic to the data of the *gas* problem.In particular estimate $y(1.75)$.

Solution

We look for a polynomial $p_2(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2$ where α_0, α_1 and α_2 are chosen so that $\sum_{i=0}^{5} (\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 - y_i)^2$ is a minimum. Differentiating with respect to α_0, α_1 and α_2 gives the three equations

$$\sum_{i=0}^{5} \left(\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2\right) = 0 \tag{4.17}$$

$$\sum_{i=0}^{5} x_i \left(\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2\right) = 0 \tag{4.18}$$

$$\sum_{i=0}^{5} x_i^2 \left(\alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2\right) = 0. \tag{4.19}$$

Inserting values for x_i and gathering coefficients leads to a 3×3 symmetric system of linear equations, namely

$$6.0\alpha_0 + 10.5\alpha_1 + 22.75\alpha_2 = 4.97 \tag{4.20}$$

$$10.5\alpha_0 + 22.75\alpha_1 + 55.125\alpha_2 = 6.855 \tag{4.21}$$

$$22.75\alpha_0 + 55.125\alpha_1 + 142.1875\alpha_2 = 12.9625. \tag{4.22}$$

Gaussian elimination gives the solution $\alpha_0 = 2.1320$, $\alpha_1 = -1.2711$ and $\alpha_2 = 0.2429$ and so the approximating quadratic is $2.1320 - 1.2711x + 0.2429x^2$.

Equations (4.20)–(4.22) are the normal equations for least squares quadratic approximation. Extending the notation used for the linear case ((4.13) and (4.14)) we may write

$$\begin{aligned}
\alpha_0 n \quad &+\alpha_1 S_x \quad +\alpha_2 S_{xx} \quad = S_y \\
\alpha_0 S_x \quad &+\alpha_1 S_{xx} \quad +\alpha_2 S_{xxx} \quad = S_{xy} \\
\alpha_0 S_{xx} \quad &+\alpha_1 S_{xxx} +\alpha_2 S_{xxxx} = S_{xxy}.
\end{aligned}$$

The pattern extends to higher order polynomials, but generally normal equations are written in the form

$$X^T X\mathbf{c} = X^T\mathbf{y}$$

which for the example above would be

$$X = \begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ \vdots & & \\ 1 & x_5 & x_5^2 \end{pmatrix}, \qquad c = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix}, \qquad y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_5 \end{pmatrix}$$

with data values $x_0 = 0.5, x_1 = 1, \ldots, x_5 = 3.0$, $y_0 = 1.62$, $y_1 = 1, \ldots, y_5 = 0.46$.

Discussion

The residuals for this approximation are shown in Table 4.11. As might be expected in nearly all cases there is an improvement over linear approximation (Table 4.10). Substituting $x = 1.75$ in the approximating quadratic we obtain the approximation $y(1.75) = 0.6512$. This value is an improvement on the linear least squares estimate of 0.8283 but still not quite in agreement with earlier estimates obtained using Newton's method, Neville interpolation and a cubic spline (0.678965, 0.678672, and 0.6812 respectively). The sum of the squares of residuals has been reduced from 0.16053 to 0.229. A least squares cubic polynomial may or may not obtain better accuracy but there is a limit as to how far the degree may be raised before the method becomes unstable and possibly produces an approximating function which oscillates wildly between the data points. Such a function could not be considered as being representative of the underlying function. In the case of the gas problem it may be that a least squares fit based on a function other than a polynomial is more appropriate. Such a function may be found by investigating the theory underlying the relation between gas and pressure or by trial and error. Approximation using the

Table 4.11 Gas problem, quadratic least squares values and residuals

y	1.62	1.00	0.75	0.62	0.52	0.46
x	0.5	1.0	1.5	2.0	2.5	3.0
quadratic least squares values	1.5571	1.1037	0.7717	0.5611	0.4720	0.5043
residuals	0.0629	−0.1037	−0.0217	0.0589	0.0480	−0.0443

least squares method is not restricted to straight lines or polynomials. Exercise 7 is one such example. The least squares method can also be used in fitting spline curves to what may be inaccurate data. In order to produce a smoothing effect we allow approximate but controlled interpolation by minimising least squares differences in higher derivatives where the spline curves meet.

Summary In this chapter we have looked at the approximation of a set of data points using a polynomial. There are two distinct possibilities here:

- determine (an explicit or implicit expression for) a polynomial that passes through all the data points (interpolation).
- determine a polynomial that may not pass through any of the data points but in some sense is a best fit.

Polynomial interpolation can conveniently be further subdivided into

- determine a polynomial interpolant with all derivatives continuous.
- determine a polynomial interpolant with only some derivatives continuous.

We have also indicated that polynomials are not the only choice. The least squares method may be used in conjunction with other functions. Where possible the choice should be guided by an understanding of the underlying model.

Exercises

1. Table 4.12 gives observed values of Young's modulus measured at various temperatures for a particular stainless steel. Use the code as part of a larger program to verify the values in Table 4.13, which show that the relation between Temperature and Young's modulus is near enough linear for practical purposes.

```
T = [0:100:600]
Y = [ 200 190 185 178 170 160 150]
% loop to find first differences
for i = 1 : 6
   diff1(i) = (Y(i+1) – Y(i))/100;  % store the first differences in diff1
end;
diff1
% loop to find the second differences
```

   Establish the normal equations for finding the coefficients a and b in the least squares straight line approximation

$$Y = a + bT$$

Table 4.12 Young's Modulus in relation to temperature

T: Temperature (Centigrade)	0	100	200	300	400	500	600
Y: Young's modulus $\times 10^{-3}$	200	190	185	178	170	160	150

Table 4.13 Differences for Young's Modulus against temperature

T	Y	1st differences	2nd differences
0	0		
		−0.1	
100	190		0.003
		−0.05	
200	185		−0.001
		−0.07	
300	178		0
		−0.08	
400	170		−0.001
		−0.1	
500	160		0
		−0.1	
600	150		

Table 4.14 Young's Modulus, least squares straight line approximation

T	Y	Least squares value
0	200	200.2500
100	190	192.2143
200	185	184.1786
300	178	176.1429
400	170	168.1071
500	160	160.0714
600	150	152.0357

to the given data, where Y is the Young's modulus ($\times 10^{-3}$) at temperature T. You may find it helpful to use Matlab expressions of the form *sum*(T) to sum the elements of a vector (or matrix) T and the $.*$ operator for element by multiplication of vectors. Transposition to either row or column form may be necessary to achieve compatibility. Solve the normal equations using the back division operator \ to show that the approximating straight line is

$$Y = 200.25 - 0.0804T. \tag{4.23}$$

Verify the results shown in Table 4.14.

This exercise is an example of *linear regression*, which is a special case of polynomial regression for which Matlab provides a function *polyfit*. Enter the command *polyfit*$(T, Y, 1)$ to check the values of b and a shown above (they will appear in this order).

2. Construct the 20 linear equations of the form $\mathbf{Ax} = \mathbf{b}$ to fit a cubic spline to the *gas* data as discussed in Sect. 4.3.4. An example of how the equations might begin to be established is shown below. In the interests of clarity leading and trailing zeros on the rows of the coefficient matrix $\mathbf{A}$ have been omitted. Solve the system using the Matlab left-division operator and so provide the coefficients of the five interpolating cubic polynomials as shown in Table 4.9. Although a 20×20 matrix to store a sparse matrix may be not the most efficient use of

resources as there are other methods for calculating cubic splines; this exercise is aimed at a primary understanding of spline interpolation.

$$\begin{pmatrix} \text{First cubic spline. y = 1.62, x = 0.5} \\ 1.0 \quad 0.5 \quad 0.25 \; 0.125 \qquad\qquad\qquad\qquad\qquad\qquad \ldots \\ \text{First cubic spline. y = 1.0, x = 1.0} \\ 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \\ \text{First and second cubic splines. Equal 1st derivatives at x = 1.0} \\ \quad -1.0 \; -2.0 \; -3.0 \; 0.0 \quad 1.0 \quad 2.0 \quad 3.0 \\ \text{First and second cubic splines. Equal 2nd derivatives at x = 1.0} \\ \qquad -2.0 \; -6.0 \; 0.0 \quad 0.0 \quad 2.0 \quad 6.0 \qquad\qquad \ldots \\ \text{First cubic spline. 2nd. derivative zero at x=0.5} \\ \qquad 2.0 \quad 3.0 \\ \text{Second cubic spline. y = 1.0, x = 1.0} \\ \qquad\qquad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \\ \text{Second cubic spline. y = 0.75, x = 1.5} \\ \qquad\qquad 1.0 \quad 1.5 \quad 2.25 \quad 3.375 \qquad\qquad \ldots \\ \text{Second and third cubic splines. Equal 1st derivatives at x = 1.5} \\ \qquad\qquad\qquad -1.0 \; -3.0 \; -6.75 \; 0.0 \; 1.0 \; 3.0 \; 6.75 \\ \text{Second and third cubic splines. Equal 2nd derivatives at x = 1.5} \\ \qquad\qquad\qquad -2.0 \; -9.0 \quad 0.0 \; 0.0 \; 2.0 \; 9.0 \\ \vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \vdots \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \\ c_0 \\ \vdots \end{pmatrix} = \begin{pmatrix} 1.62 \\ 1.0 \\ 0 \\ 0 \\ 0 \\ 1.0 \\ 0.75 \\ 0 \\ 0 \\ \vdots \end{pmatrix}$$

3. Plot the five interpolating cubic polynomials found in question 2 to show the smooth nature of the spline. An outline of the program is shown below. The program assumes that the coefficients of the cubic polynomials, as shown in Table 4.9 have been stored in the 5×4 matrix C. The program uses the Matlab function *polyval* to evaluate a polynomial at a given point.

```
for i = 1 : 5;
% Choose 20 equally spaced points in each interval, i
  xvalues = linspace( x(i),  x(i+1),  20);
  % Evaluate the polynomial at each point in the interval, i
  % polyval expects a vector of coefficients in descending
  % order, beginning with the coefficient of the highest power
  poly = C( i, : );   % select row i of C
  yvalues = polyval( poly, xvalues);
  plot ( xvalues, yvalues);
  % After plotting the first interval, plot the rest on the same graph using
  % the hold function
  if interval == 1;
     hold;
  end;
end;
```

For full details of the functions provided by Matlab for the construction and visualisation of splines refer to the Spline Toolbox.

4. Write a Matlab program to determine the coefficients of the Newton interpolating polynomial for the *gas* problem as discussed in Sect. 4.3.1. Consider using

a matrix whose (i, j) entry is to contain the value $y[x_i, \ldots, x_{i+j}]$ as given by formula (4.3). A series of single loops, of the form

```
for i = 1 : 5;
  diff(1, i) = (y(i+1)−y(i))/(x(i+1) −x(i));
end ;
for i = 1 : 4;
   diff(2, i) = (diff(1, i+1)−diff(1 ,i))/(x(i+2) −x(i));
end ;
⋮  ⋮
```

would solve the problem. However a major part of this scheme may be condensed by using *nested loops* (as shown below) following the first of the loops shown above.

```
for j=2 : 5;
  for i = 1 : 6−j;
    diff(j, i) = (diff(j−1, i+1) - diff(j−1, i)) /(x(i+j)− x(i));
  end;
end;
```

The elements of diff(1,5), diff(2,4), diff(3,3), diff(4,2), diff(5,1), ... will contain the coefficients which appear in the solution given on page 78.

5. Plot the Newton interpolating polynomial $p(x)$, which was found in the previous question. Use the following code which generates y-axis values for 40 equally spaced x-axis values in the range [0.5, 3.0].

```
xaxis = linspace( 0.5, 3.0, 40);
% evaluate the polynomial, pval at each point in the range
% using the scheme shown in Sect. 4.3.1
for i = 1 : 40
  pval= 1.62; xdiff = 1;
  for j = 1 : 5
    % use x (pressure data) from the previous question
    xdiff = xdiff∗(xaxis(i) − x(j));
    % use diff values from the previous question
    pval = pval+ xdiff∗diff(j, 1);
  end;
  yaxis(i) = pval;
end;
```

As a further exercise plot the same polynomial over an extended range for example, $0 \le xaxis \le 5$. Decide if the polynomial could be used for extrapolation based on the given data.

6. The following Matlab code will produce the values in the *linear* column of Table 4.6 using those in the *Volume* column. *xstar* (= 1.75) is the Pressure at which an estimate of the Volume is desired using the data of the *gas* problem.

```
x = 0.5 : 0.5 : 3.0;
P(: , 1) = [1.62 1 0.75 0.62 0.52 0.46]';
xstar = 1.75;
% Calculate the 'linear'  column in Table 4.6
for i = 2 : 6;
   P(i, 2) = ((xstar−x(i−1))*P(i, 1)−(xstar − x(i))*P(i−1, 1)) /(x(i)−x(i−1));
end;
P
```

Continue the program by writing a further series of loops to complete the lower triangular matrix **P** of entries shown in Table 4.6.

7. The Michaelis–Menten equation occurs in Chemical Kinetics and has the form

$$v(x) = \frac{ax}{b+x}. \tag{4.24}$$

Determine the coefficients a and b so that (4.24) is a least squares approximation to the following data obtained in an experiment to measure velocity (v) of an enzymed–catalysed reaction at various concentrations (x) (see Table 4.15).

If we were to carry out the instruction in the chapter to the letter we would minimise the sum of squares S given by

$$S = \sum_{i=0}^{7} \left(v_i - \frac{ax_i}{b+x_i} \right)^2$$

where x_i and v_i, $i = 0, 1, \ldots, 7$ are the given data points. However in this case setting partial derivatives $\frac{\partial S}{\partial a}$ and $\frac{\partial S}{\partial b}$ to zero results in non-linear equations for a and b. We could use the methods of Chap. 3 but we prefer to avoid potential difficulties by minimising the function P given by

$$P = \sum_{i=0}^{7} \big(v_i(b+x_i) - ax_i \big)^2.$$

We have

$$\begin{aligned} \frac{\partial P}{\partial a} &= -2\sum_{i=0}^{7} \big(v_i(b+x_i) - ax_i \big) x_i \\ \frac{\partial P}{\partial b} &= 2\sum_{i=0}^{7} \big(v_i(b+x_i) - ax_i \big) v_i \end{aligned}$$

and so equating derivatives to zero gives

Table 4.15 Michaelis–Menten, data

x	0.197	0.139	0.068	0.0427	0.027	0.015	0.009	0.008
v	21.5	21	19	16.5	14.5	11	8.5	7

Table 4.16 Michaelis–Menten, least squares approximation

x_i	v_i	Least squares value
0.197	21.5	21.56
0.139	21.0	20.89
0.068	19.0	18.80
0.043	16.5	16.84
0.027	14.5	14.49
0.015	11.0	11.11
0.009	8.5	8.23
0.008	7.0	7.61

$$a\sum_{i=0}^{7} x_i^2 - b\sum_{i=0}^{7} x_i v_i = \sum_{i=0}^{7} v_i x_i^2$$
$$a\sum_{i=0}^{7} x_i v_i - b\sum_{i=0}^{7} v_i^2 = \sum_{i=0}^{7} v_i^2 x_i .$$

Having established data vectors x and v use the following Matlab commands

```
A= [ x ∗ x'   −x ∗ v';   x ∗ v'   −v ∗ v']
rhs =[ v ∗ (x .∗ x)';   (v .∗ v) ∗ x']
sol = A\rhs;
```

to establish the normal equations

$$0.06577a - 9.83401b = 1.37226$$
$$9.8401a - 1989b = 197.069.$$

Solve this system using back-division and using the a and b values in (4.24). Check the validity of the approximation either by plotting the function or tabulating the result (as shown in Table 4.16).

Chapter 5
Numerical Integration

Aims In this chapter we look at ways of calculating an approximation to a definite integral of a real-valued function of a single variable, $f(x)$ (the **integrand**). Sometimes integration can be achieved using standard formulae, possibly after some manipulation to express the integral in a particular form. However, often standard formulae cannot be used (for example, the function may be known at a discrete set of points only) and so it is necessary to resort to numerical techniques. The points at which the integrand is to be evaluated are known as *grid points* or *mesh points*, although the terms *data points* or just *points* may also be used. The terms *grid* or *mesh* are equivalent and generally refer to a collection or division of an interval into *grid points* or *mesh points*.

Overview All the methods that we look at involve approximating an integral with a weighted sum of evaluations of the integrand. The methods differ in some or all of the following aspects:

- the spacing of the points at which the integrand is evaluated.
- the weights associated with the function evaluations.
- the accuracy that a method achieves for a given number of function evaluations.
- the convergence characteristics of the method (how the approximation improves with the addition of new function values).

In particular we look at

- the trapezium rule.
- Simpson's rule.
- Newton–Cotes rules.
- Gaussian quadrature.
- adaptive quadrature.

Acquired Skills After reading this chapter you will be able to estimate values of definite integrals of a wide range of functions. Although the methods which you will have at your disposal will give an approximate result, you will nevertheless be able to judge the level of accuracy. Furthermore, you will be able to make an informed choice of method to use in any given application.

5.1 Introduction

A number of standard formulae are available for evaluating, analytically, definite integrals. A particular example is provided by

$$\int_a^b x^k\,dx = \left[\frac{x^{k+1}}{k+1}\right]_a^b = \frac{1}{k+1}\left[b^{k+1} - a^{k+1}\right]$$

so that, for example, $\int_0^1 x\,dx = \frac{1}{2}$. In addition, there are a number of standard techniques for manipulating an integral so that it matches one of these formulae. A particular case in point is the technique of integration by parts:

$$\int_a^b u(x)\frac{dv(x)}{dx}\,dx = \left[u(x)v(x)\right]_a^b - \int_a^b v(x)\frac{du(x)}{dx}\,dx$$

which can be used, for example, to derive the result

$$\int_0^1 xe^x\,dx = xe^x\,\Big|_0^1 - \int_0^1 e^x\,dx = e - e + 1 = 1.$$

However, practical problems often involve an integrand (the function to be integrated) for which there is no known analytic form. The integral $\int e^{-t^2}\,dt$ is an example. On the other hand integrals may arise which involve an integrand requiring considerable effort to reduce to standard form and so a numerical approximation may be more practical. Other practical problems involve integrands for which values are only available at a number of discrete points. In this chapter we look at ways of dealing with all such problems. The first group of methods we investigate are known as the Newton–Cotes rules and include the trapezium rule, Simpson's rule and higher order rules, based on evaluating the integrand at equally-spaced points. If there is no restriction on the points at which the integrand may be evaluated, then other methods such as Gaussian quadrature may be more suitable.

5.2 Integration of Tabulated Functions

To illustrate the methods for dealing with functions for which values are only available at a number of points we consider the following problem.

Problem

A company has a contract to build a ship, the cost of which will be a function of, amongst other things, the surface area of the deck. The deck is symmetric about a line drawn from the tip of the bow to the tip of the stern (Fig. 5.1) and so it is sufficient to determine the area of one half of the deck and then to double the result. The ship is 235 metres long. Table 5.1 gives half-breadths (distances from the middle of the deck to the edge) at various grid points along the centre line, as measured from a draughtsman's drawing of the proposed vessel. Grid points are

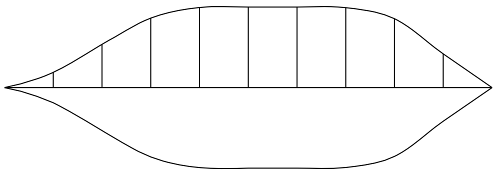


Fig. 5.1 Plan view of deck

Table 5.1 Half-breadths of the ship

Grid point	Half-breadth	Grid point	Half-breadth
0	0	6	16.36
1	3.09	7	16.23
2	8.74	8	13.98
3	14.10	9	6.84
4	16.26	10	0
5	16.36		

measured at intervals of 23.5 metres. Given this information the problem, which we subsequently refer to as the *ship builder's problem*, is to find the surface area of the deck.

5.2.1 The Trapezium Rule

The data in Table 5.1 gives some idea of the shape of the ship but is incomplete since the precise form of the hull between any two grid points is not defined. However, it is reasonable to assume that the shape is fairly regular and that whilst some curvature is likely, this is so gradual that, as a first approximation, a straight edge may be assumed. This is the basis of the trapezium rule.

Problem

Solve the ship builder's problem using the trapezium rule.

Solution

The area between grid points 0 and 1 and between grid points 9 and 10 is found by using the formula for the area of a triangle, and the area between grid points 1 and 2, 2 and 3, and so on up to 8 and 9 is found by using the formula for the area of a trapezium, namely area of a trapezium = half the sum of the parallel sides × the distance between them.

For example, the area between grid points 4 and 5 is given by $\frac{16.26+16.36}{2} \times 23.5 = 383.25$ m^2. To determine the whole surface area we simply sum the areas between consecutive grid points. The calculations (without the units) are shown in Table 5.2. The total area (of one half of the deck) is found to be 2631.06 m^2.

Table 5.2 Area of the half deck

Grid points	Area between grid points
0, 1	$\frac{3.09}{2} \times 23.5 =$ 36.31
1, 2	$\frac{3.09+8.74}{2} \times 23.5 =$ 139.00
2, 3	$\frac{8.74+14.10}{2} \times 23.5 =$ 268.37
3, 4	$\frac{14.10+16.26}{2} \times 23.5 =$ 356.73
4, 5	$\frac{16.26+16.36}{2} \times 23.5 =$ 383.29
5, 6	$\frac{16.36+16.36}{2} \times 23.5 =$ 384.46
6, 7	$\frac{16.36+16.23}{2} \times 23.5 =$ 382.93
7, 8	$\frac{16.23+13.98}{2} \times 23.5 =$ 354.97
8, 9	$\frac{13.98+6.84}{2} \times 23.5 =$ 244.64
9, 10	$\frac{6.84}{2} \times 23.5 =$ 80.37
	Total area = 2631.06

Discussion

To summarise this approach mathematically we regard Fig. 5.1 as a graph, with the x-axis running along the middle of the ship, and grid points along the x-axis. The y-axis then represents half-breadths. We let $x_i : i = 0, 1, \ldots, 10$ represent points at which half-breadths f_i are known, so that, for example, $x_2 = 47$ m and $f_2 = 8.74$ m. We let h_i be the distance between points x_i and x_{i+1} (in this example $h_i = 23.5$, $i = 0, 1, \ldots, 9$). Further, we let A_i be the area of the trapezium defined by the sides parallel to the y-axis starting at the points x_i and x_{i+1} on the x-axis. Then we have the formula

$$A_i = h_i \left(f_i + f_{i+1}\right)/2 \tag{5.1}$$

so that, for example, $A_4 = 383.29$ m^2. Hence

$$\text{Total area} = \sum_{i=0}^{9} A_i \tag{5.2}$$

which is valid even though two of the trapezia have one side of zero length (that is, the trapezia are triangles).

We can see from Table 5.2 that in the total sum each of the half-breadths corresponding to an internal section (that is, a section other than x_0 and x_{10}) is counted twice. Further, if all spacings h_i have the same value, h we have the alternative formula

$$\text{Total area} = \frac{h}{2}\left(f_0 + 2\sum_{i=1}^{9} f_i + f_{10}\right). \tag{5.3}$$

This formula is often written as

$$\text{Total area} = h \sum_{i=0}^{n}{}'' f_i, \tag{5.4}$$

where the double prime on the summation sign means that the first and last terms are to be halved.

5.2.2 *Quadrature Rules*

In the example above the triangles and trapezia are known as **panels**. Formulas such as (5.1) which are based on the areas of panels are known as **quadrature rules**. When a quadrature rule is derived by combining several simple rules, as in (5.2), (5.3) or (5.4), we refer to a **composite (quadrature) rule**. In general, a composite rule based on $n + 1$ points takes the form

$$\text{Total area} = \sum_{i=0}^{n} w_i f_i . \tag{5.5}$$

We refer to such rules, which are a linear combination of a set of **observations** f_i in which the w_i are the **weights**, as n-point (counting from zero), or n-panel, rules. The two-point rule (5.1) which determines the area of a trapezium is understandably called the **trapezium rule**. For the composite trapezium rule the weights in (5.5) are given by $w_0 = w_n = h/2$ and $w_i = h$ for $i \neq 0$ or n. Note that the term *composite trapezium rule* is often abbreviated to *trapezium rule*.

The first solution to the ship builder's problem made the assumption that it was safe to approximate the shape of the hull by straight lines. This seems reasonable, particularly in the middle section, but it is less defensible at the bow and stern.

To improve the situation (without any justification, at the moment) we employ a formula similar to (5.3), but change the weighting. This forms the basis of Simpson's Rule[1] which we illustrate in the following problem.

5.2.3 *Simpson's Rule*

Problem

Solve the ship builder's problem using Simpson's rule.

Solution

This time the end grid points are given a weight of $\frac{h}{3}$, the odd-numbered internal grid points are given a weight of $\frac{4h}{3}$, whilst the even-numbered internal grid points

[1]The rule is attributed to Thomas Simpson 1710–1761, mathematician and fellow of the Royal Society, but it was in use a hundred years earlier. Known as the Oracle of Nuneaton he was forced to flee to Derby after dressing up as the Devil.

are given a weight of $\frac{2h}{3}$, to give the formula

$$\text{Total area} = \frac{h}{3}\left(f_0 + 4\sum_{i=1}^{5} f_{2i-1} + 2\sum_{i=1}^{4} f_{2i} + f_{10}\right). \tag{5.6}$$

We then have an estimate of the total area of 2641.09 m^2, which we suggest is likely to be more accurate than that given by the trapezium rule.

Discussion

Now for a partial justification of Simpson's rule (a more comprehensive justification is given in the next section). In the previous subsection the ten-panel composite rule (5.3) was derived from the one-panel rule (5.1). Similarly, the composite rule (5.6) can be obtained by accumulating a sequence of 5 partial results. We group the grid points into sets of 3, to obtain $(0, 1, 2)$, $(2, 3, 4)$, and so on. Then for each group we compute an area estimate using the weighting $\frac{h}{3}$, $\frac{4h}{3}$ and $\frac{h}{3}$. It is the ratio $1 : 4 : 1$ which is important; the factor $\frac{1}{3}$ is there to ensure that the final calculation is scaled correctly. The ratio $1 : 4 : 1$ ensures that the formula is accurate not only for linear functions (which was the case for the trapezium rule) but (as we will see) also for quadratic and cubic polynomials and so in general is likely to provide greater accuracy. When partial results are combined to form a composite rule the ratio becomes $1 : 4 : 2 : \cdots : 4 : 2 : 4 : 1$, as in (5.6).

5.2.4 Integration from Irregularly-Spaced Data

The rules that we have used so far have been based on the assumption that the data supplied represents the outline of the hull to a reasonable degree of accuracy. Near the centre of the ship we can expect the shape between grid points to be fairly flat, and so the trapezium rule estimates based on the information supplied will be adequate. Near the bow and stern we would expect more curvature which, as we will see later, is something Simpson's rule attempts to reflect. An alternative approach is to increase the amount of information available in those regions where the curvature is most pronounced.

Problem

In order that the cost of production can be more accurately determined, the ship building company have supplied additional information in which an extra two equally spaced points have been introduced between grid points 0 and 1, 1 and 2, 8 and 9, and 9 and 10, and mid-points between grid points 2 and 3, 3 and 4, 6 and 7, and 7 and 8. Unfortunately the figures supplied by the ship building company, given in Table 5.3, failed to include the data for the point 0.25.

Table 5.3 Additional half-breadths

Section	Half-breadth	Section	Half-breadth
0.5	1.05	7.5	15.56
0.75	1.97	8.25	12.68
1.25	4.33	8.5	11.01
1.5	5.74	8.75	9.06
1.75	7.22	9.25	4.56
2.5	11.73	9.5	2.34
3.5	15.57	9.75	0.40
6.5	16.36		

Solution

Clearly, we can employ the trapezium rule between any two consecutive points provided that we take account of the fact that some are distance 23.5 m apart, others $\frac{23.5}{2}$ m apart, and yet others $\frac{23.5}{4}$ m apart. Alternatively, formula (5.5) can be used with appropriate values given to points and weights to reflect the use of this rule. Whichever way we do it, the result is that the area is computed as 2625.77 m^2.

Determining a Simpson's rule approximation using this new information presents more of a problem. We recall that the rule requires points to be grouped into sets of three which are equally spaced. Hence, we can form groups such as $(4, 5, 6)$, $(6, 6.5, 7)$, $(9, 9.25, 9.5)$ and $(9.5, 9.75, 10)$ and then apply Simpson's rule to each group. In each group we use the formula

$$A_i = \frac{h}{3}\left(f_{2i} + 4f_{2i+1} + f_{2i+2}\right) \tag{5.7}$$

where A_i is the area defined between grid points x_{2i} and x_{2i+2} having a mid-point x_{2i+1} and corresponding half-breadths f_{2i}, f_{2i+1} and f_{2i+2}, and with $h = x_{2i+1} - x_{2i} = x_{2i+2} - x_{2i+1}$.

Unfortunately, since data for the quarter-Section 0.25 was not supplied, by the time we have used $(0.5, 0.75, 1.0)$, $(1.0, 1.25, 1.5)$, $(1.5, 1.75, 2.0)$, $(2.0, 2.5, 3.0)$, …, $(8.0, 8.25, 8.5)$, $(8.5, 8.75, 9.0)$, $(9.0, 9.25, 9.5)$, $(9.5, 9.75, 10.0)$ we have a pair of points, namely $(0, 0.5)$, left over. There are various possible solutions to this problem. We could, for example, simply use the trapezium rule to approximate the area between these two points, but this is less than ideal since, as we shall see in the next section, the trapezium rule tends to produce a less accurate integral approximation than Simpson's rule. Since we are summing a number of results, the increased accuracy of Simpson's rule would be nullified.

A better, and potentially more accurate, alternative is to employ the formula

$$A_i = \frac{h}{12}\left(5f_i + 8f_{i+1} - f_{i+2}\right) \tag{5.8}$$

for determining the area of the hull between any two points x_i and x_{i+1}, with $x_{i+2} - x_{i+1} = x_{i+1} - x_i = h$. It can be shown that formula (5.8), which uses a value external to the interval in question, has an accuracy closer to Simpson's rule than the trapezium rule.

If, in addition to the conventional Simpson's rule, the points $(0, 0.5, 1)$ are used in (5.8) we obtain an area estimate for the half deck of 2631.06 m^2, which we expect to be the most accurate so far.

5.3 Integration of Functions

We now generalise our earlier observations and produce quadrature rules for finding the areas bounded by a function $f(x)$, the x-axis, and two lines parallel to the y-axis. In particular we reintroduce, in a more formal manner, the trapezium and Simpson's rules outlined in the previous section and justify some of the statements we have previously made about these formulae.

5.3.1 Analytic vs. Numerical Integration

Inevitably, there are some integrals which are impossible, or at best difficult, to evaluate analytically, and we are then obliged to employ an appropriate numerical technique which produces an integral estimate. In fact nearly all the integrals likely to be met in the course of solving practical problems are of this form. The basis of these techniques is the implicit construction of a function $\hat{f}(x)$ which approximates $f(x)$ and can be easily integrated. This usually involves a discretisation using a set of grid points $\{x_i \ : \ i = 0, 1, \ldots, n\}$, with $x_i < x_{i+1}$, at which values $f_i = f(x_i)$ are known or can be calculated. The integral estimate then takes the form of a quadrature rule (5.5).

In this section we are concerned with the evaluation of the integral of f over the interval $[a, b]$, written $\int_a^b f(x)\,dx$. We assume that $f(x)$ is a continuous function of the independent variable x and that we are able to evaluate $f(x)$ (the integrand) at any point in $[a, b]$. We assume that $x_0 = a$, $x_n = b$, and that, unless otherwise stated, all grid points are equally spaced, distance h apart.

5.3.2 The Trapezium Rule (Again)

One of the simplest forms of quadrature rule is based on the approximation of f within each subinterval $[x_i, x_{i+1}]$ by a straight line; that is, within $[x_i, x_{i+1}]$ we have

$$\hat{f}(x) = -\frac{1}{h}(x - x_{i+1}) f_i + \frac{1}{h}(x - x_i) f_{i+1} \tag{5.9}$$

where $h = x_{i+1} - x_i$ and $i = 0, 1, \ldots, n - 1$.

The fact that $\hat{f}(x_i) = f_i$ and $\hat{f}(x_{i+1}) = f_{i+1}$ can be readily verified by substituting x_i and x_{i+1} in turn into (5.9). We say that $\hat{f}(x)$ **interpolates** $f(x)$ at the points x_i and x_{i+1}. Because of its straight-line form, $\hat{f}(x)$ is referred to as the **linear interpolant** and the form given here is equivalent to that given by (4.1) of Chap. 4 with f replacing y and x replacing $\hat{x}$.

Since $x_0 = a$ and $x_n = b$, we have

$$\int_a^b f(x)\,dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)\,dx.$$

But

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \int_{x_i}^{x_{i+1}} \hat{f}(x)\,dx$$
$$= -\frac{1}{h}\int_{x_i}^{x_{i+1}} (x - x_{i+1}) f_i\,dx + \frac{1}{h}\int_{x_i}^{x_{i+1}} (x - x_i) f_{i+1}\,dx.$$

Now

$$\int_{x_i}^{x_{i+1}} (x - x_{i+1}) f_i\,dx = f_i \int_{-h}^{0} x\,dx = -\frac{h^2}{2} f_i$$

and similarly

$$\int_{x_i}^{x_{i+1}} (x - x_i) f_{i+1}\,dx = f_{i+1} \int_{0}^{h} x\,dx = \frac{h^2}{2} f_{i+1}.$$

Hence we have replaced an integral that, in principle, we were unable to perform analytically by the sum of two integrals which are readily determined analytically. Hence we obtain the approximation

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx \frac{h}{2}(f_i + f_{i+1})$$

which compares with (5.1).

Summing over all the grid points we have

$$\int_a^b f(x)\,dx \approx h \sum_{i=0}^{n}{}'' f_i.$$

It can be shown that if I_i represents the true integral and R_i the approximation over a single panel $[x_i, x_{i+1}]$, then the error, $E_i = I_i - R_i$ is given by

$$E_i = -\frac{h^3}{12} f''(\eta_i) \tag{5.10}$$

where η_i is some (unknown) point between x_i and x_{i+1}.

The proof depends on writing the difference between $f(x)$ and the linear interpolant $\hat{f}(x)$ as

$$f(x) - \hat{f}(x) = \frac{(x - x_i)(x - x_{i+1})}{2} f''(\Upsilon)$$

where Υ is some point in $[x_i, x_{i+1}]$. The error of the interpolation, E_i is given by

$$E_i = \int_{x_i}^{x_{i+1}} \frac{(x - x_i)(x - x_{i+1})}{2} f''(\Upsilon)\, dx.$$

This integral cannot be evaluated directly since Υ is a function but we can use the second mean value theorem for integrals to show

$$E_i = \frac{f''(\eta_i)}{2} \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\, dx$$

where η_i is in the range $[x_i, x_{i+1}]$ and so by carrying out the integration we have (5.10).

For the composite rule R, the error, $E = I - R$, where I is the true integral over $[a, b]$, consists of the sum of individual one-panel errors of the form (5.10). This can be simplified to

$$E = -\frac{h^2}{12}(b - a) f''(\eta) \tag{5.11}$$

where η is some point in $[a, b]$. The result is interesting. The second derivative of a straight line function is zero, but for a quadratic it is non-zero; hence the trapezium rule integrates a straight line exactly (as we would expect) but not a quadratic. The other point to note is that since E is proportional to h^2 then a reduction in the mesh spacing by a factor of 2 (i.e. doubling the number of panels) should, theoretically, reduce the error by a factor of 4 (although this argument only holds for sufficiently *smooth* functions).

5.3.3 Simpson's Rule (Again)

Simpson's rule follows in a similar way, using approximation by quadratics instead of approximation by straight lines (cf. Sect. 5.2.3). We expect such an approximation to be more accurate since it allows for a certain amount of curvature. We take an odd number of grid points $\{x_i \; : \; i = 0, 1, \ldots, 2n\}$ (and hence even number of panels) with $x_0 = a$ and $x_{2n} = b$.

We require an approximating quadratic $\hat{f}(x)$ such that $f(x_j) = \hat{f}(x_j)$ for $j = 2i, 2i + 1, 2i + 2, i = 0, 1, \ldots, n - 1$ and for this it is convenient to use the Lagrange interpolation formula namely,

$$\begin{aligned}\hat{f}(x) = \frac{1}{2h^2}\big((x - x_{2i+1})(x - x_{2i+2}) f_{2i} - 2(x - x_{2i})(x - x_{2i+2}) f_{2i+1} \\ + (x - x_{2i})(x - x_{2i+1}) f_{2i+2}\big)\end{aligned} \tag{5.12}$$

which is Newton's interpolating quadratic, (4.5) Chap. 4, in a different form. We then make the approximation

$$\int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \approx \int_{x_{2i}}^{x_{2i+2}} \hat{f}(x)\,dx.$$

To integrate (5.12) we follow the pattern set by our analysis of the trapezium rule. We have

$$\begin{aligned}\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i+1})(x - x_{2i+2})\,dx &= \int_{-h}^{h} x(x-h)\,dx \\ &= \left[\frac{x^3}{3} - h\frac{x^2}{2}\right]_{-h}^{h} \\ &= \frac{2h^3}{3}.\end{aligned}$$

Similarly

$$\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+2})\,dx = -\frac{4h^3}{3} \tag{5.13}$$

and

$$\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i-1})\,dx = \frac{2h^3}{3}. \tag{5.14}$$

Aggregating these results we obtain the approximating rule

$$\int_{x_{2i}}^{x_{2i+2}} f(x) \approx \frac{h}{3}\left(f_{2i} + 4f_{2i+1} + f_{2i+2}\right). \tag{5.15}$$

Summing all approximations of this form we have

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1} \int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \\ &= \frac{h}{3}\left(f(a) + 4\sum_{i=1}^{n-1} f(x_{2i-1}) + 2\sum_{i=1}^{n-1} f(x_{2i}) + f(b)\right).\end{aligned}$$

As with the trapezium rule it is possible to obtain an expression for the error in the Simpson's integration rule.

We have that for a single application

$$E = -\frac{h^5 f^{(\mathrm{iv})}(\eta)}{2880} \tag{5.16}$$

and for the composite rule

$$E = -\frac{h^4 f^{(\mathrm{iv})}(\eta)}{180}(b-a) \tag{5.17}$$

where η is some unknown point between a and b.

Since the fourth derivative of a cubic polynomial is identically zero, we have a rule that despite being constructed to integrate a quadratic, Simpson's rule will integrate a cubic exactly.

Table 5.4 Projected sales using the trapezium rule

Number of grid points	Sales	Error
33	810 096	6531
65	805 199	1634
129	803 973	408
257	803 666	101
513	803 590	25
1025	803 571	6
2049	803 566	1
4097	803 565	0

Since E is proportional to h^4, a halving of the mesh spacing will, theoretically, reduce the error by a factor of 16. For most integrals, therefore, we would expect Simpson's rule to give a much better approximation than the trapezium rule, and we give an example of such behaviour in the next section.

Before leaving this section we provide a justification for formula (5.8). The rule is derived by integrating the Lagrange interpolation formula (5.12) over a single panel $[x_i, x_{i+1}]$ (rather than over two panels as in Simpson's rule). This rule does not have the same accuracy as Simpson's rule. A single application gives $E \propto h^4 f^{(\mathrm{iii})}(\eta)$ for some η; that is, the rule does not integrate a cubic exactly. However, the rule is potentially more accurate than the trapezium rule and hence is to be preferred.

Finally we use the following problem, henceforth to be referred to as the *Sales* problem, to compare Simpson's rule and the trapezium rule.

Problem

A manufacturing Company is about to launch its new product. After extensive market research the company has forecast that, using suitable promotion strategies, over the next year world-wide sales will grow exponentially; starting at day 0, sales on day x will be $e^{x/36.5} - 1$. The company wants to know what the total sales will be over the first year.

Solution

Although the problem has been described in discrete form (that is, sales per day) we can regard it as a continuous problem in which we need to determine I, the area under the graph of $f(x) = e^{x/36.5} - 1$ over the interval $[0, 365]$. This is a simple integration which can be performed analytically; $\int (e^{x/36.5} - 1)\, dx = 36.5e^{x/36.5} - x$ and so $I = 36.5(e^{10} - e^0) - 365 \approx 803\,565$ to the nearest whole number.

Table 5.4 gives results for trapezium rule estimates to the integral (again, to the nearest whole number) using various numbers of grid points. Table 5.5 shows equivalent results obtained using Simpson's rule.

Table 5.5 Projected sales using Simpson's rule

Number of grid points	Sales	Error
9	812 727	9162
17	804 215	650
33	803 606	41
65	803 567	2
129	803 565	0

Discussion

Since we know the true answer we can compute the actual error in the approximation rules. For the trapezium rule the reduction in the error by a factor of roughly four when the number of intervals is doubled is clearly demonstrated in Table 5.4. For Simpson's rule the results (Table 5.5) exhibit a reduction in the error by a factor of about 16 when the number of intervals is doubled. Tables 5.4 and 5.5 also show just how more accurate Simpson's rule is compared to the trapezium rule for all numbers of intervals. Another way of expressing this is that Simpson's rule can yield the same accuracy as the trapezium rule using far fewer grid points. If we take the number of function evaluations required to compute an approximation as a measure of the efficiency of the two methods, then clearly Simpson's rule is the more efficient.

5.4 Higher Order Rules

We have established a pattern of approximation by a straight line followed by approximation by a quadratic. This is a concept which is easy to extend to approximations by a cubic, a quartic and still higher orders. It might be thought that this leads to improved quadrature rules, but unfortunately this is not necessarily so. If we use a cubic $\hat{f}(x)$ for approximation purposes, so that $f(x_i) = \hat{f}(x_i)$ for $i = 0, 1, 2, 3$, we obtain the (three-eighths) formula

$$\int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}\big(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\big). \tag{5.18}$$

It transpires that the error, E in using the three-eighths over an interval $[a, b]$ is proportional to $(b-a)^5 f^{(\mathrm{iv})}$, and so the rule is no better than Simpson's rule (although we might employ it, in conjunction with the composite Simpson's rule, if we were given an even number of grid points).

The type of rules we have been investigating are collectively known as Newton–Cotes[2] rules. However there are good numerical reasons why extending the

[2]Roger Cotes, 1682–1716. Mathematician and Astronomer who worked closely with Newton. However there was some doubt as to his commitment. According to fellow astronomer Halley he missed a total eclipse being 'opprest by too much company'.

Newton–Cotes type rules that we have been investigating is not necessarily productive. Using approximating polynomials of higher and higher order does not necessarily improve the corresponding quadrature rule approximation to the integral. The weights become large and alternate in sign, leading to potential problems with rounding error. Further, even if we were able to employ exact arithmetic, there is no guarantee that a sequence of Newton–Cotes approximations of increasing order will converge to the true integral. It is rare, therefore, that one is ever tempted to use any rule of the form considered here other than those we have defined. Simpson's rule or the three-eighths rule is about as far as we would go. If high accuracy is required then it is best to use a low order rule and reduce the mesh spacing and/or switch to a rule for which the grid points are not equally spaced.

5.5 Gaussian Quadrature

The rules we have considered so far have all been based on a set of grid points which are equally spaced, but there is no particularly good reason for this (other than this may be the only form of data available). If the integrand can be sampled at any point within $[a, b]$ there are better choices for the x_i. We assume, without loss of generality, that $[a, b] = [-1, 1]$. It is a simple matter to employ a linear transformation to map any other interval onto $[-1, 1]$; for $x \in [a, b]$ we form $t = (2x - a - b)/(b - a)$ and then $t \in [-1, 1]$. Hence $\int_a^b f(x)\,dx = \frac{b-a}{2}\int_{-1}^{1} f(((b - a)t + a + b)/2)\,dt$.

In the formula $\sum_{i=0}^{n} w_i f_i$, first seen as (5.5), we now choose n and the points and weights according to Table 5.6. These values ensure that a rule with n points integrates exactly a polynomial of degree $2n - 1$ (as opposed to n for n odd and $n + 1$ for n even, as is the case with a Newton–Cotes rule). There is no simple relationship between the error and the mesh spacing since the latter is no longer constant. It can be observed that the points are distributed symmetrically about $[-1, 1]$ (and the weights are similarly distributed) and that, apart from 0, no two rules share common grid points. Further, all weights for all Gaussian quadrature formulas are positive.

It may be shown that the Gauss points are the roots of the Legendre polynomials $P_n(x)$ which are defined by the three-term recurrence relation

$$P_{n+1}(x) = \frac{2n + 1}{n + 1} x P_n(x) - \frac{n}{n + 1} P_{n-1}(x) \tag{5.19}$$

with $P_0(x) = 1$ and $P_1(x) = x$. Using (5.19) we have that $P_2(x) = \frac{3}{2}x^2 - \frac{1}{2} = \frac{1}{2}(3x^2 - 1)$. The roots of $P_2(x)$ are, therefore, $\pm\frac{1}{\sqrt{3}} = \pm 0.5773502692$.

Having established the Gauss points corresponding weights are determined by the following system of equations.

$$\int_{-1}^{1} x^j\,dx = \frac{1}{j + 1}\left[x^{j+1}\right]_{-1}^{1} = \sum_{i=0}^{n} w_i x_i{}^j, \quad j = 0, 1, \ldots, 2n - 1. \tag{5.20}$$

Table 5.6 Gauss points and weights for $[-1, 1]$

Number of grid points	Points	Weights
2	±0.5773502692	1.0
3	±0.7745966692	0.5555555556
	0.0	0.8888888889
4	±0.8611363116	0.3478548451
	±0.3399810436	0.6521451549
5	±0.9061798459	0.2369268850
	±0.5384693101	0.4786286705
	0.0	0.5688888889

This system ensures that weights are chosen so that the Gauss rule integrates a polynomial of degree $2n - 1$ exactly. Although the system of equations (5.20) is linear in the weights, it is nonlinear in the points, and so a nonlinear solver, such as Newton's method (Chap. 2) must be used.

Problem

Solve the *Sales* problem using Gaussian quadrature.

Solution

Transforming the range of integration from $[0, 365]$ to $[-1, -1]$ using the substitution $t = (2x - 365)/365$ we have

$$\int_0^{365} \left(e^{x/36.5} - 1\right) dx = \frac{365}{2} \int_{-1}^{1} \left(e^{5(1+t)} - 1\right) dt.$$

Applying the Gaussian five-point rule we use the formula

$$I = \frac{365}{2} \sum_0^4 w_i \left(e^{5(1+t_i)} - 1\right)$$

to find an estimate, I of the integral. The t_i and w_i are the Gauss points and corresponding weights for five grid points shown in Table 5.6. The result is shown in Table 5.7.

By way of comparison we also produce an estimate based on dividing the range $[0, 365]$ into $[0, 365/2]$ and $[365/2, 365]$ producing and then adding estimates of the integral over each range. Following transformations to the range $[-1, 1]$ the relevant integrals are

$$\frac{365}{4} \int_{-1}^{1} \left(e^{2.5(1+t)} - 1\right) dt \quad \text{and} \quad \frac{365}{4} \int_{-1}^{1} \left(e^{2.5(3+t)} - 1\right) dt.$$

Using a Gaussian five-point rule in each range we obtain the further result shown in Table 5.7. The increase in accuracy for a given number of function evaluations

Table 5.7 Projected sales using Gaussian quadrature

Number of Gauss points	Sales	Error
5	803 210	355
10	803 563	2

over both the trapezium rule and Simpson's rule (Tables 5.4 and 5.5) approximations becomes evident.

5.6 Adaptive Quadrature

Our approach to quadrature has so far been one of selecting an appropriate rule and then choosing enough grid points to ensure that the integral estimate so obtained is sufficiently accurate. Unfortunately we have, as yet, no mechanism for deciding just how many points are required to yield the required accuracy. This number will depend critically on the form of the integrand; if it oscillates rapidly over $[a, b]$, then a large number of points will be required; if it varies only very slowly, then we can make do with a small number of points. We need a measure of this variation which can be used to concentrate the points at appropriate sub-regions in $[a, b]$.

The strategy, which may be applied to any of the quadrature rules considered previously, may be expressed in general terms as

1. Obtain a first integral approximation R_1 using $n + 1$ grid points.
2. Obtain a second integral approximation, R_2, using $2n + 1$ grid points.
3. Using R_1 and R_2 obtain an estimate of the error in R_1.
4. If the error estimate is small enough accept R_1 (or R_2).
5. Otherwise split $[a, b]$ into two equal subintervals and repeat the above process over each.

This is a recursive definition; the strategy is defined in terms of itself. We illustrate the method by applying it to the trapezium rule.

Problem

Use the trapezium rule in an adaptive manner to estimate $\int_0^1 e^x\, dx$, for which the exact integral is $e - 1$, (1.718282).

Solution

We obtain a first approximation to the integral I using the one-panel trapezium rule. This leads to the approximation, R_1 with error E_1 where

$$R_1 = \frac{1}{2}\left(e^0 + e^1\right) = 1.8591. \tag{5.21}$$

Table 5.8 Adaptive trapezium method

Original interval and sub-intervals	Estimate	Error (absolute values)	Current eps	Eps less than 0.01
0 . . . 1	1.7189	0.140280	0.01	No
0 . . . 0.5	0.6487	0.013445	0.005	No
0 . . . 0.25	0.2840✓	0.001477	0.0025	Yes
0.25 . . . 0.5	0.3647✓	0.001897	0.0025	Yes
0.5 . . . 1	1.0696	0.022167	0.005	No
0.5 . . . 0.75	0.4683✓	0.002436	0.0025	Yes
0.75 . . . 1	0.6013	0.003128	0.0025	No
0.75 . . . 0.875	0.2819✓	0.000367	0.00125	Yes
0.875 . . . 1	0.3194✓	0.000416	0.00125	Yes
Sum of ✓ estimates	1.7183			

Using the two-panel trapezium rule gives a further approximation, R_2 with error E_2.

$$R_2 = \frac{1}{4}\left(e^0 + 2e^{0.5} + e^1\right) = 1.7539. \tag{5.22}$$

We know from (5.10) that the error, E_1 in the approximation R_1 is given by

$$E_1 = I - R_1 = kh^3 \tag{5.23}$$

where k has a value which depends on the value of the second derivative of the integrand at some point in [0, 1]. In this first approximation $h = 1$, but we will continue to use the variable name h to illustrate the general method. In considering further approximations to I we make the simplifying assumption that the second derivative of the integrand is constant over the range of integration and so we have two contributions to the error term, each with half the interval spacing to give

$$E_2 = I - R_2 = 2k(h/2)^3. \tag{5.24}$$

Eliminating k from (5.23) and (5.24) we obtain

$$I = \frac{1}{3}\,(4R_2 - R_1) \tag{5.25}$$

which in the example we are using gives $I = 1.7189$ which is closer to the accurate integral (1.7182) than the previous estimate. In practice it is the error estimates that are of interest and eliminating I from (5.23) and (5.24) gives $kh^3 = \frac{4}{3}(R_2 - R_1) = -0.1403$.

In order to integrate over an interval $[a, b]$ we continually subdivide intervals in which the error term is greater than a predefined tolerance. If the overall tolerance was set to *eps*, tolerances within each subdivided interval would be successively halved. The final result would be formed by adding the results from each subinterval.

Table 5.8 illustrates the procedure using the current example with *eps* set to 0.01. Adding the estimates for the 'Yes' rows, the ticked ✓ estimates for intervals [0, 0.25], [0.25, 0.5], [0.5, 0.75], [0.75, 0.875] and [0.875, 1], gives a total estimate of 1.7183 for the integral, which is correct to the accuracy shown. The adaptive method applies equally to other composite rules. Matlab implements the adaptive version of Simpson's method in the *quad* function.

Summary Whilst some integration problems that arise in practice may be solved using analytic techniques, it is often the case that some numerical method has to be used to derive an approximate solution. Knowing that a numerical solution is expressed in terms of an integration rule, which is a weighted sum of evaluations of the integrand, two questions arise:

- Which rule should be used?
- How many function evaluations will the rule involve?

We have the following rules

- Newton–Cotes rules.
  - the trapezium rule.
  - Simpson's rule.
- Gauss rules.

In general, for the same number of function evaluations we can expect Simpson's rule to give a better approximation than the trapezium rule, and a 3-point Gauss rule to give even better results again. It will always be possible to construct integrands for which these conclusions are not true, but that should not act a deterrent unless there is evidence to show that the task in hand is not feasible. Increasing the number of points in a rule may give improved accuracy. When choosing the number of function evaluations consideration must be given to the accuracy required. Adaptive quadrature is a technique for choosing this number automatically and is therefore often the more attractive proposition.

Exercises

1. Write a Matlab function *trap* with the heading

$$\mathit{function}[\mathit{defintegral}] = \mathit{trap}(x, f, n)$$

to implement the trapezium rule. The function is to return an approximation to the integral of the function f(x) over the range $[a, b]$ based on n equally spaced grid points stored in the vector **x** with $x(1) = a$, $x(n) = b$ and with corresponding function values stored in the vector **f**. Use a *for* loop within the *trap* function to evaluate the sum shown as (5.4).

Establish the code in an M-file *trap.m* and test the function using a program to compute a standard integral, for example

```
x = linspace( 0, 1, 2 );
y = x;
integ = trap( x, y, 4 )
```

which returns in variable *integ* the Trapezium approximation (in this case an accurate evaluation) to $\int_0^1 x\,dx$ based on 2 grid points.

2. Evaluate $\int_0^1 x^2\,dx$ using function *trap* from the previous question with different numbers of grid points. In each case calculate the difference between the correct value $(1/3)$ and the estimated value. Confirm that the difference is as predicted by (5.11). Use the formula to predict (and confirm by running the program) that a mesh of 14 points would be required for absolute error of less than 0.001.
3. Modify the Matlab code for *trap* to produce a function *Simpson* with a similar heading. Use a formula based on a more general form of (5.6) to implement the method. Assume an odd number of grid points. Ensure that the input parameter n is an *odd* number by using the Matlab function *floor*, which rounds down to the nearest integer. Use separate loops for the *odd* and *even* sums.

```
function [integral] = Simpson( x, f, n )
% Assume function values for the range of integration are stored in
% equally spaced values x(i), y(i) i=1..n where is an odd number
% if n is even print an error message and return to the calling program
  if 2∗ floor( n/2 ) == n;
  disp('Error:number of grid points must be odd');
  return;
else
  %
  % Insert the Simpson code
  %
end;
```

Use the function *Simpson* to show that $\int_0^1 x^2\,dx$ and also $\int_0^1 x^3\,dx$ are evaluated correctly and that the error in evaluating $\int_0^1 x^4\,dx$ are all as predicted by (5.17).
4. Compare the rates at which the Simpson's rule and the trapezium rule converge to an estimate of

$$\int_0^1 \frac{1}{1+x^2}\,dx$$

with increasing number of grid points. The accurate value is $\pi/4$.

Note that as with element by element multiplication using the `.*` dot–asterisk operator, Matlab provides a corresponding element by element division operator `./` dot–slash operator and so an x, y mesh of n evenly spaced points spanning the interval [0, 1] may be generated using the following commands:

```
x = linspace( 0, 1, n );
y = 1 ./ sqrt(1 + x.∗x);
```

Use the Matlab commands *trapz* to apply the trapezium rule and *quad* based on Simpson's rule to check the results from your own routines. For the example above, assuming x and y and a function $f(x)$ to evaluate $1/(1+x^2)$ were already defined, suitable commands would be

```
trapz(x, y)
quad(@f,0,1)  % use the @ prefix to pass a function name as a parameter
```

In general $trapz(x, y)$ integrates y values over corresponding x values and $quad(@f, a, b)$ integrates the function f between the limits a and b. By default *quad* approximates to within an accuracy of $1.0e^{-6}$ and requires the function $f = f(x) = 1/(1 + x^2)$, where x is the vector of data points, to be defined in an M-file with corresponding name $f.m$

5. Write a Matlab function with the heading *function*[*points*] = *Gmesh(n)* to generate the n roots of the Legendre Polynomial $P_{n-1}(x)$. Using recurrence relation (5.19) we have $P_0 = 1$, $P_1 = x$ and $P_2 = \frac{1}{2}(3 * x^2 - 1)$ and so since we are writing the coefficients of the first three Legendre Polynomials as Matlab vectors for use in the function *roots* to find the roots, we have

```
p = [1];
q = [1 0];
r = (3/2)∗ [q 0] - (1/2)∗[0 0 p];
```

It would be possible to continue in this way using a new variable name at each step. Using a matrix is a possibility but there are complications in that each row regarded as a vector would have one more element that the one before. A convenient option is provided by the *cell* structure (question 1 of Sect. 1.15.2) as shown below:

```
p(1) = {[1]}
p(2) ={[1 0]};
```

Using (5.19) and noting that we are storing the coefficients of the ith polynomial in $p(n)$ we have the Matlab expression

$$p(i+2) = \{((2*i+1)/(i+1)) * [p\{i+1\}\ 0] - (i/(i+1)) * [0\ 0\ p\{i\}]\};$$

which may be incorporated in a *for* loop (from 1 to $n - 2$) within the function *Gmesh*. The final action of *Gmesh* would be to call *roots* with input parameter $\{p\}$ to find the roots of $p(n)$. Test the function against the values shown in Table 5.6.

6. Write a program to calculate the weights corresponding to the relative Gauss points. Equation (5.20) shows that for the first two Legendre Polynomials, P_1 and P_2 the linear systems to be solved are:

$$\begin{aligned} w_1 + w_2 &= 2 \\ w_1 + w_1 x_2 &= 0 \\ (x_1 = 0.5774, x_2 &= -0.5774) \end{aligned} \qquad \begin{aligned} w_1 \quad + w_2 \quad + w_3 &= 2 \\ w_1 x_1 + w_2 x_2 + w_3 x_3 &= 0 \\ w_1 x_1 + w_2 x_2^2 + w_3 x_3^2 &= \tfrac{2}{3} \\ (x_1 = 0, x_2 = 0.7746, x_3 = -0.7746). \end{aligned}$$

In deriving the second system from formula (5.20), note that $0^0 = 1$. It follows that in the second example the leading coefficient of the first equation in this example will be one (and not zero). The general pattern for forming a general $n \times n$ system may be deduced. In forming the coefficient matrix, the Matlab operator $.\wedge$ for raising the elements of a vector (or matrix) to a given power (for example $v = v. \wedge 2$ or $e = g. \wedge j$) may be useful. Test the program against the values shown in Table 5.6.

7. Use the Gauss 5-point rule to estimate the integral $I = \int_0^{\frac{\pi}{2}} \sin x \, dx$. In order to use the rule transform the range of integration from $[0, \pi/2]$ to $[-1, +1]$ by making the substitution

$$t = -1 + \frac{4}{\pi} x$$

so that when $x = 0$, $t = -1$ and when $x = \frac{\pi}{2}$, $t = +1$. We now have

$$I = \frac{\pi}{4} \int_{-1}^{+1} \sin \frac{\pi}{4} (t + 1) \, dt.$$

Use the Gauss 5-point rule

$$I = \frac{\pi}{4} \sum_{i=0}^{4} w_i \sin \frac{\pi}{4} (p_i + 1)$$

where p_i and w_i, $i = 1, \ldots, 5$ are the Gauss points. The accurate value, which may be found analytically is 1.0.

8. Modify the function *trap* developed earlier to produce a new function *Etrap* which for a given function *f* returns an estimate of the integral $\int_a^b f(x) \, dx$ using the trapezium rule and in the notation of Sect. 5.6 the error quantity kh^3. The estimates are to be based on using just 2 and 3 grid points as described in the solution to the problem shown in Sect. 5.8.

   Use the following heading for the function

$$function[integral, error] = Etrap(a, b, f)$$

and store the program in a file *Etrap.m*. The program will need to call the original *trap* function first with parameter n set to 2 and again with $n = 3$.

   Test the program using the example quoted in Sect. 5.6 and a command such as *Etrap*(0, 1, @*exp*) or [*int*, *err*] = *Etrap*(0, 1, @*exp*); to return the values for the estimated integral and the estimated error in variables *int* and *err* respectively.

9. As an introduction to the next question. Write a function *fact* with heading

$$function[Factorial] = fact(n)$$

to evaluate the factorial of an integer n, usually denoted by n! Use the principle of *recursion* by which a function is evaluated by repeatedly evaluating simpler versions of itself until a stage is reached (a *stopping condition*) at which a value of the function is known. For this example a stopping condition would be $1! = 1$. Following the header, a single statement of the following form will suffice

```
if n== 1;
  Factorial = 1; % stopping condition
else
  Factorial = n * fact(n-1);
```

10. Write a Matlab function *Atrap* which for a given function f uses Adaptive Quadrature based on the Trapezium Rule to return an estimate of the integral $\int_a^b f(x)\,dx$ to within a prescribed accuracy *eps*. Use the following heading for the function

$$function[integral] = Atrap(a, b, f, eps)$$

and base the program on the following recursive scheme:

```
% estimate the integral over the range [a, b] using Etrap (question 8)
  [i e] = Etrap (a, b, f); % (for example)
% is the absolute value of the estimated error returned by Etrap less than eps?
% if it is
  integral = i; % the stopping condition
% if not, halve the interval, halve the required accuracy and try again,
% by treating the two
% intervals separately. Keep on doing this as long as is necessary.
  eps = eps/2;
  integral = Atrap(a, (a+b)/2, f, eps)+ Atrap((a+b)/2, b, f, eps);
```

Test the program against standard integrals with known values, for example use the command

```
Atrap(0,1,@exp,1.0e-006)
```

to validate the results shown in Table 5.8. You may like to check the progress of the recursion by inserting statements to display the intervals and to show how they are subdivided to achieve the required accuracy.

Chapter 6
Numerical Differentiation

Aims In this chapter we look at numerical methods for estimating derivatives of a function $f = f(x)$. We establish formulae for evaluating $f'(x)$ and higher derivatives for any value of x.

Overview The methods of this chapter are not intended to be used on a function for which values are only available at specific points. Although the underlying function in question may be analytic, the methods assume that we can provide function values wherever we are asked to do so. If the precise form of the function is not known it is better to construct an approximation using the method of least squares or to construct a spline approximation in the manner discussed in Chap. 4 and use the resulting function to estimate derivatives. In this way a global view of the function is taken and so is likely to produce more accurate estimates than a method which relies on piecemeal interpolation to supply individual values.

We point out that the methods of this chapter are also not intended to be used where expressions for derivatives may be obtained without too much difficulty using the rules of differential calculus. However the functions we use in illustrating the methods will be mainly of this type as they are useful for verification purposes. Of course, any analytic function may be differentiated, but in some cases, for example in the case of complicated rationals, the actual differentiation is not for the faint hearted. In any event Matlab has facilities for displaying derivatives in symbolic (algebraic) form, which will be explained in an end-of-chapter exercise. Estimates for derivatives are often required in solving differential equations for which an analytic solution is not available. If a numerical solution is therefore the only option, progress to a solution may be made by replacing derivatives by estimates based on function values

The methods to be described fall into two classes. On the one hand we derive formulae based on function values in the vicinity of the point at which we wish to estimate the derivative. In this context we derive two, three and five-point formulae based on two, three and five local points respectively. Our second approach is to transform the problem of estimating the derivative into that of estimating the value of an integral, a problem that has already been considered in Chap. 5.

Acquired Skills After reading this chapter you will be able to

- recognise the circumstances in which numerical differentiation rather than analytic differentiation may be usefully applied.
- understand how two, three and five-point formulae are derived.
- apply two, three and five-point formulae to finding estimates of first and higher order derivatives.
- understand how Cauchy's theorem can be used to transform the problem of estimating derivatives into that of estimating an integral from which we may obtain estimates for first and higher order derivatives.

6.1 Introduction

As is often the case in numerical analysis we take Taylor's theorem as our starting point. For a function f which is continuous and has continuous derivatives in an interval $[a, b]$ for any points x and $x + h$ in that interval, we have formulae such as

$$f(x+h) = f(x) + hf'(x) + R_1 \tag{6.1}$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + R_2 \tag{6.2}$$

$$\vdots \quad \vdots$$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \cdots + \frac{h^m}{m!}f^{(m)}(x) + R_m. \tag{6.3}$$

In each case the remainder term R_i depends on the next higher derivative. In particular

$$R_m = \frac{h^{m+1}}{(m+1)!}f^{(m+1)}(\psi_m) \tag{6.4}$$

where ψ_m is some point in the interval $(x, x + h)$.

6.2 Two-Point Formula

We can use expressions (6.1)–(6.3) to provide formulae for numerical differentiation. For example the expression (6.1) may be written in the form

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{1}{h}R_1 \tag{6.5}$$

which leads to the approximation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{6.6}$$

The approximation (6.6) is known as a *two-point formula*, since it uses function values at two points to estimate the derivative.

Table 6.1 Estimates of $f'(0)$ for $f(x) = \sin(x)$

n	$f'(0)$	Error
1	0.9589	0.0411
2	0.9896	0.0104
3	0.9974	0.0026
4	0.9993	0.0007
5	0.9998	0.0002
6	1.0000	0.0000
7	1.0000	0.0000

Problem

For $f(x) = \sin(x)$ estimate $f'(x)$ at $x = 0$.

Solution

In using (6.6) we have no indication as to an appropriate value for h other than it should be *small* to reduce the remainder term R_1. Accordingly we start with $h = 0.5$ and successively half the value until a consistent value for f' is found. Putting $x = 0$ in (6.6) we obtain the results (see Table 6.1) for successive values of h, where $h = \frac{1}{2^n}$, $n = 1, 2, \ldots$.

Discussion

The results show that (to four decimal places) the value of the derivative, $f'(0)$ is 1.0000 and so the numerical approximation agrees with the analytical result ($f'(0) = \cos(0) = 1$). For $f(x) = \sin x$, $f''(x) = -\sin x$ and so $|f''(x)| \leq 1$. This implies that the remainder term R_1, which can be found from (6.4) with $m = 1$, is at most $\frac{h}{2}$ whatever the value of ψ_1, which in turn guarantees the accuracy of the estimate for sufficiently small h. Furthermore the error, namely $\frac{R_1}{h}$ (the absolute value is shown) is halved as h is halved. This is a result to be expected from (6.4). Even though in this case the final value h may appear to be small, even smaller values may be necessary as the following example shows.

Problem

For $f(x) = x^5$ estimate $f'(x)$ at $x = 2$.

Solution

Using (6.6) we obtain the results in Table 6.2.

Table 6.2 Estimates of $f'(2)$ for $f(x) = x^5$

n	$f'(2)$	Error
1	131.3	51.3
2	102.7	22.7
3	90.65	10.65
4	85.16	5.16
$\vdots$	$\vdots$	$\vdots$
13	80.01	0.01
14	80.00	0.00
15	80.00	0.00

Discussion

In this case $f''(x) = 20x^3$ which can become quite large for $|x| > 1$. It follows that the remainder term is no longer exclusively controlled by the size of h and the guarantee which we had for $f(x) = \sin(x)$ no longer applies. The sequence above indicates that we need to choose a much smaller value of h (2^{-15}) in order to achieve a reliable estimate. Again it can be seen that the error, for which the absolute value is shown, decreases linearly with h.

In theory it seems that as long as the second derivative is bounded we could use the two-point formula (6.6) to estimate the first derivative of any function $f(x)$ provided we make h sufficiently small. However, given the way that computing systems perform their arithmetic it is not always possible to do this. In attempting to evaluate

$$\frac{f(x+h) - f(x)}{h}$$

for a given x and h, where h is *small*, we may be asking the computer to divide one small number by another. Inevitably these numbers will be subject to error and when they are close to rounding error we will have problems. For example, what in theory may be $\frac{10^{-6}}{10^{-7}}$, namely 10, might be distorted to $\frac{10^{-6}-10^{-8}}{10^{-7}+10^{-8}}$ and computed as 9. For this reason we would avoid the two-point formula if it became apparent that such a situation might occur.

6.3 Three- and Five-Point Formulae

From (6.2) we have

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(\xi) \tag{6.7}$$

and

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(\eta) \tag{6.8}$$

where ξ is some point in $(x, x+h)$ and η is some point in $(x-h, x)$. Subtracting (6.8) from (6.7) to remove the unwanted second derivative, we have

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \frac{h^2}{12}\big(f'''(\xi) + f'''(\eta)\big) \tag{6.9}$$

and so we have the approximation

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}. \tag{6.10}$$

The approximation (6.10) is known as a *three-point formula*, even though the mid-point $f(x)$ is missing. There are three-point formulae which explicitly involve three points, but these tend to be slightly less accurate than the one we have given here. Assuming the third derivative to be bounded, the error in the approximation is of order h^2, which is one order higher than for the two-point formula and hopefully should lessen the need for very small h values.

Problem

Consider the function $f(x) = x^2 \sin(1/x)$, $x \neq 0$ and $f(0) = 0$. Obtain estimates for $f'(x)$ at $x = 0.1$ using a two-point formula and a three-point formula.

Solution

We use (6.6) and (6.10). As in the previous example we begin with $h = 0.5$ and then successively halve the value until consistent values for f' are found. We obtain the results in Table 6.3 for successive values of h, where $h = \frac{1}{2^n}$, $n = 1, 2, \ldots$. Both formulae produce the value of the derivative, correct to four decimal places.

Discussion

This example illustrates that we can expect more rapid convergence if we base the estimate of the derivative on a larger number of points. Roughly speaking, the error in using the two-point formula decreases by a factor of 1/2 in the later stages as h is halved, whereas for the three-point formula the error decreases by a factor 1/4.

By using Taylor's theorem we can produce formulae for estimating $f'(x)$ using any number of nearby points, such as x, $x+h$, $x+2h$, $x-h$, and so on, for a suitably small h. All such formulae may be derived in the same way as (6.10) but rather than produce formulae just for the sake of it we quote one particular five-point formula which is considered to be very useful.

$$f'(x) \approx \frac{1}{12h}\big(f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)\big). \tag{6.11}$$

Table 6.3 Estimates of $f'(0.1)$ for $f(x) = x^2 \sin(1/x)$

	$f'(0.1)$			
n	Two-point formula	Error	Three-point formula	Error
1	0.7267	0.0027	0.4541	0.2762
2	0.1593	0.5710	0.0856	0.6447
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
10	0.7632	0.0329	0.7289	0.0013
11	0.7470	0.0168	0.7299	0.0003
12	0.7387	0.0085	0.7302	0.0001
13	0.7345	0.0043	0.7302	0.0000
14	0.7324	0.0021	0.7302	0.0000
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
18	0.7304	0.0001		
19	0.7303	0.0001		
20	0.7303	0.0000		
21	0.7303	0.0000		

The remainder term in (6.11) is proportional to h^4 and the value of the fifth derivative at some point in the range $(x - 2h, x + 2h)$ and so, all being well, the approximation should offer better prospects for convergence than the two and three point formulae. Rapid convergence offers a better hope of avoiding the very small values of h, which we know can cause trouble.

Problem

Using the previous example $f(x) = x^2 \sin(1/x)$, $x \neq 0$ and $f(0) = 0$, calculate estimates for $f'(x)$ at $x = 0.1$ using the five-point formula (6.11).

Solution

As before we choose $h = \frac{1}{2^n}$, $n = 1, 2, \ldots$ and obtain the results in Table 6.4.

Discussion

The value -0.0372 of the second line of the table is somewhat surprising and serves as a warning that formulae for numerical differentiation should not be applied in isolation but rather as part of an iteration. As expected the five-point formula converges more rapidly than the three-point formula. However, at each step the five-point formula uses twice as many function evaluations as the three-point formula. As already indicated this is about as far as we intend to go. For most practical applications estimates of first derivatives may be obtained by using a two-point formula with sufficiently small h. However, if that value of h becomes *too* small for the system to

Table 6.4 Further estimates of $f'(0.1)$ for $f(x) = x^2 \sin(1/x)$

$f'(0.1)$	
n	Five-point formula
1	0.3254
2	−0.0372
⋮	⋮
9	0.7302
10	0.7303
11	0.7303

perform reasonably accurate calculations we would opt for a three-point formula, with possibly a five-point formula as a last resort.

6.4 Higher Order Derivatives

We could use Taylor's theorem to derive formulae for higher derivatives but instead we use the formulae we have already obtained. For example, from (6.6) we have

$$f''(x) \approx \frac{f'(x+h) - f'(x)}{h}$$

and in conjunction with the original (6.6) this provides a means of estimating $f''(x)$. We can use the same method to estimate $f'''(x)$ and even higher derivatives.

Similarly we may use the more accurate three-point formula (6.10) to obtain

$$f''(x) \approx \frac{f'(x+h) - f'(x-h)}{2h}. \tag{6.12}$$

A further application of (6.10) produces $f''(x)$ in terms of $f(x)$ as can be seen in the following example.

Problem

Given $f(x) = x^5$ use the formula (6.12) to estimate $f''(1)$.

Solution

From (6.12) with $h = 0.5$ we have

$$f''(1) \approx f'(1.5) - f'(0.5). \tag{6.13}$$

Table 6.5 Estimates of $f''(1)$ for $f(x) = x^5$

$f''(0.1)$	
n	Three-point formula
1	30
2	22.5
$\vdots$	$\vdots$
6	20.01
7	20.00
8	20.00

Using (6.10) to estimate $f'(1.5)$ and $f'(0.5)$ we have (with $h = 0.5$)

$$\begin{aligned} f'(1.5) &\approx f(2) - f(1) \\ &= 31 \\ f'(0.5) &\approx f(1) - f(0) \\ &= 1. \end{aligned}$$

Substituting these results in (6.13) gives $f''(1) \approx 30$. As before we halve the value of h ($h = \frac{1}{2^n}$, $n = 1, 2, \ldots$) and repeat the calculation and continue in this manner until a consistent result emerges. The results are summarised in Table 6.5. We conclude that $f''(1) = 20.00$, which is the expected result.

Discussion

The approximation used in solving this problem may be derived more formally. Substituting for $f'(x)$ from (6.10) in (6.12) we have

$$\begin{aligned} f''(x) &\approx \frac{f'(x+h) - f'(x-h)}{2h} \\ &\approx \frac{f(x+2h) - 2f(x) + f(x-2h)}{4h^2}. \end{aligned} \tag{6.14}$$

It is usual to write (6.14) in the form

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}. \tag{6.15}$$

This approximation may also be obtained directly from Taylor's theorem.

6.4.1 Error Analysis

It should be appreciated that in estimating second derivatives using (6.12), and using the same h, we are dividing by h^2 rather than h, as was the case with the first derivative. In the case of third, fourth and higher derivatives we are dividing by h^3, h^4 and so on. Moreover as we use a formula such as (6.12) to progress from one order of derivative to the next we lose an order of accuracy since the original error term is also divided by h. Given the previous warnings regarding h there is even more danger in allowing h to become too small when estimating higher derivatives. It has to be emphasised that although in theory a smaller h should lead to greater accuracy, in practice this is not necessarily so.

Problem

Use the five-point formula (6.11) to estimate the first five derivatives of the following function $f(x)$ at $x = 0$

$$f(x) = \frac{e^x}{\sin^3 x + \cos^3 x}.$$

This function is often used for test purposes since it is awkward to differentiate by hand and therefore represents the class of functions which we might well choose to differentiate numerically. Also it can be shown that $f^{(v)}(0) = -164$, which is a useful benchmark.

Solution

From (6.11) with $h = 0.5$ we have

$$\begin{aligned} f'(0) &= \frac{1}{6}\big(f(-1) - 8f(-0.5) + 8f(0.5) - f(1)\big) \\ &= 0.6258. \end{aligned}$$

To estimate $f''(0)$, using the same value of h we again use (6.11) but in the form

$$f''(0) = \frac{1}{6}\big(f'(-1) - 8f'(-0.5) + 8f'(0.5) - f'(1)\big). \tag{6.16}$$

The values of f' necessary to evaluate (6.16) are found by applying (6.11) in its original form. As can be seen this is quite a lengthy process, particularly by the time we reach $f^{(v)}$. However, the use of recursion reduces the programming effort considerably.

As usual we halve the value of h ($h = \frac{1}{2^n}$, $n = 1, 2, \ldots$) and repeat the calculation and continue in this manner until a consistent result emerges. The results are summarised in Table 6.6.

Discussion

We conclude that $f'(0) = 1$, $f''(0) = 4$, $f'''(0) = 4$ and $f^{(\mathrm{iv})}(0) = 28$. As for $f^{(\mathrm{v})}(0)$ we cannot be so sure. The results do not settle down in the manner of the lower derivatives and it is with less certainty that we deduce the result to be approximately -164. Letting the calculation proceed with smaller and smaller values of h would not help. We would find that the estimates become larger and larger

Table 6.6 Estimates of high derivatives of $f(x) = e^x/(\sin^3 + \cos^3 x)$ at $x = 0$

n	$f'(0)$	$f''(0)$	$f'''(0)$	$f^{(\text{iv})}(0)$	$f^{(\text{v})}(0)$
1	0.6258	1.1089	−3.0587	80.3000	−50.6253
2	1.0391	−4.0883	−103.4769	307.0102	4025.6974
3	1.0015	3.9993	4.9547	310.3380	13035.8616
4	1.0001	3.9999	4.0209	27.8954	−161.7975
5	1.0000	4.0000	4.0013	27.9938	−163.8619
6	1.0000	4.0000	4.0001	27.9996	−163.9914
7			4.0000	28.0000	−163.9995
8			4.0000	28.0000	−164.0000

as the computational inaccuracies begin to dominate. It seems that for this particular function we have pushed our methods to the limit in attempting to estimate the fifth derivative. Any attempt to calculate higher derivatives for this and similarly-complicated functions may produce estimates which do not converge.

Although it might be possible to derive estimates for higher and higher derivatives we have reached the end of the road as far as the present approach is concerned. We have provided reliable methods for estimating low-order derivatives which in all probability is all that will be required in practice. It is difficult to imagine a modelling situation requiring fourth and fifth, let alone sixth or even higher derivatives! However, the problem is not impossible and we indicate a method based on Cauchy's theorem using complex variables.

6.5 Cauchy's Theorem

Cauchy's theorem states that

$$\frac{f^{(n)}(\psi)}{n!} = \frac{1}{2\pi i}\int_C \frac{f(z)}{(z-\psi)^{n+1}}\,dz, \quad n = 1, 2, \ldots \tag{6.17}$$

where $f(z)$ is differentiable in a neighbourhood which includes a closed contour C which surrounds the point ψ.

Using results from the theory of complex variables, the details of which need not concern us, we can show that (6.17) may be transformed to

$$\frac{f^{(n)}(\psi)}{n!} = \frac{2}{r^n}\int_0^1 f\left(re^{2\pi i t} + \psi\right)\cos(2\pi n t)\,dt, \quad n = 1, 2, \ldots \tag{6.18}$$

where we disregard the *imaginary* part of the integrand, and where r is the radius of a circle, centre ψ in which $f(z)$ is analytic in the complex plane. We can say that a function $f(z)$ will be analytic if it can be differentiated and the value of the differential does not tend to infinity. However, this still leaves the problem of deciding upon a suitable value for r when using (6.18). We show in the following examples how this difficulty may be overcome.

Table 6.7 Cauchy estimates of $f'(0)$ for $f(x) = e^x/(\sin^3 + \cos^3 x)$

$f'(0)$	
r	Estimate using Cauchy
1.0	1.0
0.5	1.0
0.1	1.0

Problem

For $f(x) = \sin(x)$ estimate $f'(x)$ at $x = 0$ using Cauchy's theorem.

Solution

In this example $\psi =$ and $n = 1$ and so making the substitutions as in (6.18) we have

$$f'(0) = \frac{2}{r}\int_0^1 \sin\big(re^{2\pi it}\big)\cos(2\pi t)\,dt. \tag{6.19}$$

Initially we set $r = 1$ and evaluate the integral using Simpson's rule (based on 51 grid points). When evaluating the integrand we separate the *real* and *imaginary* parts, and discard the latter. We obtain the results in Table 6.7. Further subdivision does not change the results, which suggests that we are already inside a circle, centre $z = 0$ in which $f(z) = \sin(z)$ is analytic in the complex plane. We therefore accept the estimate $f'(0) = 1.0$, which is also the correct value. In this particular case, since $f(z) = \sin z$ is analytic everywhere in the complex plane it is hardly surprising that we readily find a suitable r. The next problem repeats an earlier example.

Problem

Use Cauchy's theorem to estimate the first five derivatives of $f = f(x)$ at $x = 0$, where

$$f(x) = \frac{e^x}{\sin^3 x + \cos^3 x}.$$

Solution

Making the appropriate substitutions as in (6.18) we have

$$\frac{f^{(n)}(0)}{n!} = \frac{2}{r^n}\int_0^1 \frac{e^{re^{2\pi it}}}{\sin^3(re^{2\pi it}) + \cos^3(re^{2\pi it})}\cos(2\pi nt)\,dt, \quad n = 1, 2, \ldots, 10. \tag{6.20}$$

As before we initially set $r = 1$ and then make reductions. The results shown in Table 6.8 agree with our previous results and those that may be obtained by direct differentiation. The integration was performed using an adaptive implementation of Simpson's method as implemented by Matlab in the function *quad* with tolerance set to 10^{-10}.

Table 6.8 Cauchy estimates of high derivatives of $f(x) = e^x/(\sin^3 + \cos^3 x)$ at $x = 0$

r	$f'(0)$	$f''(0)$	$f'''(0)$	$f^{(\text{iv})}(0)$	$f^{(\text{v})}$
1.0	1.5634	2.7751	8.1846	8.2401	−44.3003
0.5	1.0000	4.0000	4.0000	28.0000	−164.0000
0.25	1.0000	4.0000	4.0000	28.0000	−164.0000

Table 6.9 Cauchy estimates of higher derivatives

r	$f^{(\text{vi})}(0)$	$f^{(\text{vii})}(0)$	$f^{(\text{viii})}(0)$	$f^{(\text{xi})}(0)$	$f^{(\text{x})}(0)$
1.0	−821.7125	−5639.8581	−30559.4056	26369.1145	2621923.0835
0.5	64.0000	−13376.0000	47248.0000	−858224.0000	13829823.9999
0.25	64.0000	−13376.0000	47248.0000	−858224.0001	13829824.0049
0.125	64.0000	−13376.0001	47247.9994	−858206.0423	13829824.9559
0.0625	63.9999	−13375.9982	47248.3478	−858247.0216	13817427.3193

Discussion

The results for higher derivatives are tabulated in Table 6.9 and show that care has to be taken when deciding on a suitable value of r for the radius of circle centred on the point at which we require the derivative. In theory r set to some very small value would suffice unless the derivative does not exist or is infinite at or near the point in question, in which case we could not expect an answer. However, computationally this is not always feasible since at some stage we are dividing by r^n. For small r rounding error could lead to a loss in accuracy. The results in the table above show this effect. However, the method is a good one and given appropriate software it is recommended for the estimation of higher derivatives.

Summary In this chapter we have considered two classes of numerical method for the estimation of derivatives of a function. We began by using Taylor's theorem to derive two, three and five-point formulae for the evaluation of the first derivative of a function $f = f(x)$ at any point x. All such formulae involve the use of a parameter h to determine points adjacent to x at which function values are required. We showed by example that the choice of h is quite crucial and for this reason it is recommended that estimates using more than one value of h should be obtained before any final decision can be made. Having established formulae for $f'(x)$ in terms of values of $f(x)$ we showed how the same formulae can be used in a recursive manner to obtain estimates of $f''(x)$ and higher derivatives.

We indicated the limitations of two, three andfive-point formulae when it comes to obtaining higher derivatives of complicated functions, but rather than produce seven-point, nine-point and even higher order formulae we presented an alternative approach based on Cauchy's theorem. Although Cauchy's theorem involves the use of complex numbers and we had to assume that computer software is in place

Table 6.10 Exercise 1 results

n	$f''(0.1)$
1	0.4537
2	1.5823
3	−0.8171
4	−3.7819
5	10.014
⋮	⋮
9	70.108
10	70.096
11	70.096

to perform complex arithmetic, the method has its attractions. In transforming the problem to that of evaluating an integral we were able to use the well-established methods of Chap. 5. The main problem is to find a circle in the complex plane centred on the point in question within which the function is analytic. Our approach was to perform the integration using a number of radii r until we could be sure that we were obtaining results within such a circle.

Although the use of Cauchy's theorem is more cumbersome to implement it would appear to offer better estimates in the case of higher derivatives. If just first or second derivatives are required the use of two, three and five-point formulae is recommended, with the five-point formula being the best option for fast convergence.

Exercises

1. Write a Matlab program to estimate $f''(0.1)$ for $f(x) = x^2 \sin \frac{1}{x}$ using the five-point formula in the form

$$f''(x) = \frac{1}{12h}\big(f'(x-2h) - 8f'(x-h) + 8f'(x+h) - f'(x+2h)\big)$$

   using the five-point formula (6.11) to evaluate values of f required in the above. Two functions, held in their respective M-files could be used, a function to evaluate $f'(x)$ that uses another to evaluate $f(x)$. Verify that results similar to results in Table 6.10 are obtained for successively smaller values of h ($h = \frac{1}{2^n}$, $n = 1, 2, \ldots$). In this example it has been necessary to reduce h to very small proportions in order to achieve a reliable estimate. The remainder term in the Taylor series approximation which underlies formula (6.10) depends on h and on the value of third derivative of the function close to the point (0.1) in question. Since the value of the third derivative is of order 10^4, the value of h has to be reduced considerably from an initial 0.5 to compensate and so produce an acceptable remainder (error) term.
2. It can be shown that the two-point formula (6.6) is *exact* for linear functions in that the formula when applied to functions such as $f(x) = ax + b$ gives accurate results whatever the value of the step-length h. Similarly the three-point formula

(6.10) is exact for quadratic polynomials and the five-point formula (6.11) is exact for quartic polynomials. Write a program to check these statements by evaluating $f'(x)$ for (i) a linear function using the two-step formula, (ii) a quadratic polynomial using the three-step formula and (iii) a quartic polynomial using the five-point formulae. In each case use arbitrary values for x, step-length h and for function coefficients.

3. It is possible to produce formulae for estimating derivatives based on values to one side of the point in question. Using techniques similar to those we have already discussed it can be shown that

$$f'(x) \approx \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$$

and

$$f'(x) \approx \frac{-25f(x) + 48f(x+h) - 36f(x+2h) + 16f(x+3h) - 3f(x+4h)}{12h}$$

are alternative three- and five-point formulae. Show by experiment that in general these formulae are not quite as effective as their more balanced counterparts (6.10) and (6.11) and so should only be used when values of the function f are not available at one side or the other of the point in question.

4. This exercise aims to estimate the derivatives of

$$f(x) = \frac{e^x}{\sin^3 x + \cos^3 x}$$

at $x = 0$ and so confirm the results quoted in the chapter.

(i) Write a Matlab function *chy* with heading

$$\textit{function}\,[\textit{chyval}] = \textit{chy}(t)$$

to evaluate the integrand of (6.20) for any given value of t and constant values r and n. Note that Matlab has facilities for handling complex (*imaginary*) numbers including arithmetic involving combinations of real and complex numbers. In general the complex number $a + ib$ (a, b real) may be written as complex (a, b) or a + bi (it is important to place i next to b and to separate the sign). The usual arithmetic operations apply.

The variables r and n, are to be declared as *global* since they are to be given values from the program which will use *chy* and so the function might have the form

```
function[chyval] = chy(t)
  global n r;
  c= r * exp(2*pi*complex(0, 1) * t);
  % using the .* and .∧ element by element multiplication and
  % exponentiation operators
  chyval = exp(c).*cos(2*pi*n*t) . / (sin(c).∧ 3 + cos(c).∧3);
```

Having stored the function in an M-file *chy*, test the function using the program

```
global n r
  r = 1; n =1;
  chy(0.5);
```

to confirm that

$$\left(\frac{e^{e^{\pi i}}}{\sin^3(e^{\pi i}) + \cos^3(e^{\pi i})}\right)\cos(\pi) = 0.8397.$$

(ii) Evaluate the integral

$$\frac{f^{(n)}(0)}{n!} = \frac{2}{r^n}\int_0^1 \left(\frac{e^{re^{2\pi it}}}{\sin^3(e^{\pi i}) + \cos^3(e^{\pi i})}\right)\cos(2\pi nt)\,dt, \quad n = 1, 2, \ldots, 10$$

using the Matlab function *quad* with tolerance 1.0e-11. Express the real part of the integrand using the function *real* as in the following code

```
c= r∗ exp(2∗pi∗complex(0, 1)∗t);
% Recover the real part of a complex number
rp = real (exp(c).∗cos(2∗pi∗n∗t) . / (sin(c) .∧ 3 + cos(c) .∧3))
```

5. The Matlab Symbolic Math Toolbox provides tools for solving and manipulating symbolic (i.e. algebraic) expressions, including the function *diff* for differentiating. Use the following program to evaluate symbolically the first, second, third, fourth and fifth derivatives of the function f at the point $x = 0$.

```
syms f d x; % declare f d and x to be symbolic variables
f = exp(x)/(sin(x)∧3 + cos(x)∧3);
x = 0;
d = diff(f); % differentiate f symbolically
eval(d) % evaluate the first differential at x = 0
for i = 1 : 4
  d = diff(d) % evaluate next differential from the one before
  eval(d) % evaluate at x = 0
end;
```

The symbolic expressions for the derivatives may be seen by removing the trailing semi-colons but not surprisingly they are quite lengthy. The function *ezplot* is available for plotting symbolic expressions. Enter the command *help ezplot* for further details.

Chapter 7
Linear Programming

Aims In this chapter we

- explain what is meant by a linear programming problem.
- show how linear programming problems arise in the real-world.
- show how a simple linear programming problem may be solved by graphical methods

and then, taking the lead from this graphical approach, devise

- the simplex method for solving linear programming problems.

Finally we consider

- the 'travelling salesman problem' as it is traditionally known: deciding the shortest route through a number of cities

and

- the 'machine scheduling problem': completing in the shortest possible time a series of processes competing for shared resources.

Overview Linear programming is concerned with finding the maximum value of a real-valued linear function f of a number of variables. We may also be asked to find the minimum value of f, but this is equivalent to finding the maximum value of $-f$ and so there is no additional problem. Such a linear function might, for example, be given by $f = f(x_1, x_2)$, where $f = x_1 + 2x_2$, although in practice many more variables are usually involved. The function in question is known as the **objective function**. Additionally, the dependent variables (x_1, x_2 in the case quoted above), are required to satisfy a number of linear relations, known as **constraints**. Thus a linear programming problem might be to find values of x_1, x_2 that

$$\begin{aligned}\text{maximise}\quad & z = x_1 + 2x_2\\ \text{subject to}\quad & 3x_1 + 2x_2 \le 5\\ & x_1 + 3x_2 \le 6\\ & x_1, x_2 \ge 0.\end{aligned}$$

The solution to a linear programming problem is known as the **optimal** solution. We may further impose the condition that some or all of the variables may only take integer values at the optimal solution. This leads to a branch of linear programming known as **integer programming**, or **mixed integer programming** (if not all variables are so constrained). A byproduct of integer programming enables us to model the decision making process when faced with alternative strategies in the search for an optimum solution.

It is worth remarking that the term *linear programming* has no connection with computer programming as such. Linear programming began life in earnest during World War II when it was used by the military in making decisions regarding the optimum deployment of equipment and supplies.

Acquired Skills After reading this chapter and completing the exercises you will be able to

- recognise if a problem is suitable for linear programming.
- model the problem in a standard form.
- solve a two variable problem using graphical methods.
- use the simplex method to solve a more general problem.
- use the branch and bound method to solve an integer programming problem (and a mixed integer programming problem).
- set up linear programming models incorporating decision making.

7.1 Introduction

Problems from the real world rarely present themselves in a form which is immediately amenable to linear programming. It is often the case that simplifications have to be made in order that the objective function and the constraints may be represented as linear functions. Even if a problem is recognised as being suitable for linear programming, data must be collected and tabulated. The skill in successful linear programming lies in the modelling of the problem and in the interpretation of the consequent solution. The calculations involved in finding the solution to a linear programming problem follow a well-defined path, which may be best left to a computer.

7.2 Forming a Linear Programming Problem

By way of an example we consider a problem that we refer to as the *garment factory* problem. Although only two variables are involved, it is indicative of the way in which larger problems are modelled.

Table 7.1 Data from the garment factory

	All-weather	Luxury-lined	Time available
Cutting	8	5	42
Sewing	3	16	60
Packing	5	6	32
Profit	500	400	

Problem

A garment factory produces two types of raincoat, the all-weather and the luxury-lined. The work involves three processes namely, cutting the material, sewing, and packing. The raincoats are made in batches but because of local conditions (availability of qualified staff, etc.) the number of hours available for each process is limited. In particular 42 hours are available for cutting, 60 hours for sewing and 32 hours for packing. Moreover the time taken to complete each process varies according to the type of raincoat. The material for a batch of all-weather raincoats takes 8 hours to cut whereas for a batch of luxury-lined only 5 hours are required. The figures for sewing a batch of all-weather raincoats and a batch of luxury-lined are 3 hours and 16 hours respectively. Similarly the figures for packing all-weather and luxury-lined are 5 hours and 6 hours respectively. The profit to be made on a batch of all-weather raincoats is £500 and on a batch of luxury-lined is £400.

The factory owner wants to know how best to use the resources available in order to maximise the daily profit. Formulate this problem as a linear programming problem. Assume that all workers are paid the same, whether they are working or not, that raw material costs and overheads are the same whatever the process, and that the factory can sell all that it produces.

Solution

To formulate the linear relationships of the problem we tabulate the data as dimensionless quantities (Table 7.1) and assign variables to the quantities involved.

Let x_1 be the number of batches of all-weather raincoats that the factory produces in a day and x_2 be the corresponding number of luxury-lined. The objective function z, in this case the profit to be made, is then given by

$$z = 500x_1 + 400x_2.$$

The constraints arise from the time available for cutting, sewing and packing. Taking each of these activities in turn we have

$$\begin{aligned} 8x_1 + 5x_2 &\le 42 \quad \text{(cutting)} && (7.1)\\ 3x_1 + 16x_2 &\le 60 \quad \text{(sewing)} && (7.2)\\ 5x_1 + 6x_2 &\le 32 \quad \text{(packing).} && (7.3) \end{aligned}$$

The first of these inequalities expresses the fact that the total time taken to cut the material for x_1 batches of all-weather raincoats ($8x_1$ hours) plus the time for cutting the material for x_2 batches of luxury-lined raincoats ($5x_2$ hours) must not exceed 42 hours. The other two inequalities are constructed on similar lines. We also add the so-called **trivial constraints**

$$x_1 \geq 0 \tag{7.4}$$

$$x_2 \geq 0 \tag{7.5}$$

since we are not interested in negative values.

Discussion

We now have a linear programming problem since the objective function and the constraints are linear relationships. It remains to find values x_1 and x_2 which maximise z. In suitably small linear programming problems a graphical method of solution may be employed as shown in the following problem.

Problem

Solve the *garment factory* problem using a graphical approach.

Solution

Since the solution which maximises z must satisfy the constraints (7.1)–(7.5) we know that the optimal solution must lie within the area bounded by the lines

$$\begin{aligned} 8x_1 + \ 5x_2 &= 42 \\ 3x_1 + 16x_2 &= 60 \\ 5x_1 + \ 6x_2 &= 32 \end{aligned}$$

and the x_1 and x_2 axes. The area in which the solution must lie is known as the **feasible region** and this is shown as the shaded region of Fig. 7.1.

Having established the feasible region we find the optimal solution by observing how the value of the objective z, given by $z = 500x_1 + 400x_2$, changes value across the region for varying x_1 and x_2. Figure 7.1 shows lines corresponding to $z = 2000$ and $z = 4000$. The first line crosses the feasible region, the other does not. If we were to draw lines corresponding to $z = 2000$, $z = 2100$, $z = 2200$ and so on, we would be drawing lines parallel to $z = 2000$ that gradually move in the direction of $z = 4000$. At some point we would be about to leave the feasible region and this defines the solution we are looking for. The (x_1, x_2) coordinates of the point that the line touches the feasible region is the optimal solution since we have found the maximum possible value of z consistent with the constraints. It is clear from Fig. 7.1 that A is the point in question and either by reading the point from the graph or (more accurately) solving the equations

$$\begin{aligned} 8x_1 + 5x_2 &= 42 \\ 5x_1 + 6x_2 &= 32 \end{aligned}$$

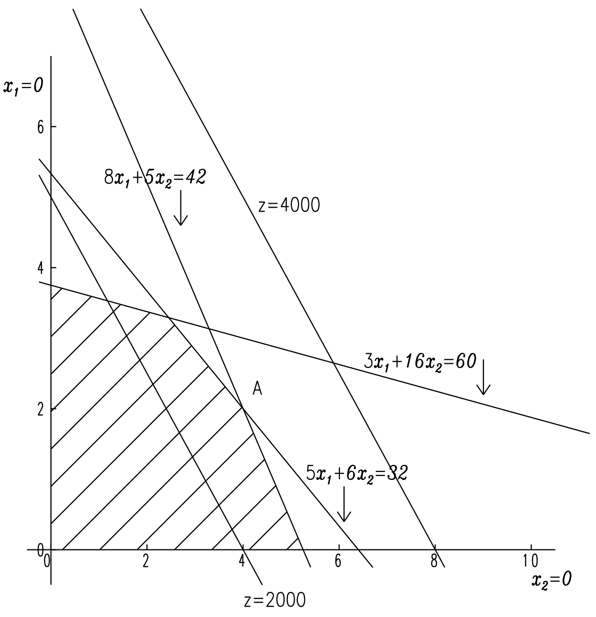


Fig. 7.1 Feasible region for the *garment factory* problem

we find that the optimal solution is $x_1 = 4$, $x_2 = 2$ and the corresponding value of z is 2800. Therefore in order to maximise profit the garment factory should produce 4 batches of all-weather raincoats and 2 batches of luxury-lined raincoats every day.

Discussion

This problem raises a number of issues in relation to its formulation, its solution and the method by which the solution is obtained.

As it has been presented here the problem bears only a passing resemblance to the real-world situation. In practice it is likely that there will be a number of different pay grades and a number of products. The cost of raw materials and the overheads for different products will not be the same. There may be limits to the number of garments of a particular type that can be sold. These and many other considerations may affect the overall profit. The number of variables involved in a realistic model of a real-world situation can become very large indeed. Problems involving thousands of variables are not uncommon. But if all the various constraints can be expressed as linear relationships and there is no inherent contradiction in the way the problem is constructed, we can use linear programming to find the optimal solution.

It will not have gone unnoticed that we obtained an integer solution, to the *garment factory* problem, namely 4 batches of all-weather raincoats and 2 batches of luxury-lined. If the solution had not arrived as whole numbers it might not have been as useful. For various reasons it might not be possible to resume work the following day on an incomplete batch. Even if this were possible we would have to consider the worth of our solution in relation to its impact on subsequent production. If an integer solution is the only one that is acceptable we can force the issue by using integer programming, which is to be discussed later in the chapter.

Given the way in which we found the solution to our problem, by moving a straight line across the feasible region, it is to be expected that the solution is found at a corner (or **vertex**) of the feasible region. This principle extends to more complex problems. If we have a problem involving three variables we would have a feasible region in the form of a three-dimensional shape made up of flat (two-dimensional) surfaces. If this is the case we can imagine the objective function as a plane moving through the polyhedral figure and reaching its maximum value at a vertex. We have theorems which state that if the feasible region of a linear programming problem is bounded then the feasible region has a polyhedral shape and that if a linear programming problem has an optimal solution then the optimal solution is found at a vertex of the feasible region.

It might appear to be a simple matter to evaluate the objective at every feasible vertex and note where the highest value of the objective occurs. However, whilst such an approach would work for small problems, it becomes totally impractical for larger problems. Even to identify the feasible region of a linear programming problem having just 16 problem variables and subject to 20 non-trivial constraints would involve examining the 7,000 million or so points at which both trivial and non-trivial constraints intersect. In any event such a problem is very small by linear programming standards. Clearly a more subtle approach is required.

However, the graphical approach has served to indicate the way ahead using a method based on the vertices of the feasible region. In the following we adopt an algebraic approach, but in order to do this we require linear programming problems to be expressed in a common form.

7.3 Standard Form

For a linear programming problem involving n variables $x_1, x_2, \ldots, x_n$ we may write the objective z as

$$z = c_1x_1 + c_2x_2 + \cdots + c_nx_n.$$

In the language of linear programming the variables $x_1, x_2, \ldots, x_n$ are known as **problem variables** and the coefficients $c_1, c_2, \ldots, c_n$ are known as **costs**. Since the problem of maximising z is the same as that of minimising $-z$ we assume as a first step towards standard form that the problem is always one of finding a set of values of the problem variables which maximises the objective. If necessary we replace each c_i by $-c_i$ and ignore any fixed price (constant overhead) costs.

As a further step towards standardisation we express all constraints (other than the trivial constraints) using the $\leq$ sign. Again, this may be achieved by changing signs of the appropriate coefficients.

It follows that any linear programming problem may be expressed in the form

$$\begin{array}{ll} \text{maximise} & z = c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{subject to} & c_{11}x_1 + c_{12}x_2 + \cdots + c_{1n}x_n \le b_1 \\ & c_{21}x_1 + c_{22}x_2 + \cdots + c_{2n}x_n \le b_2 \\ & \vdots \qquad\qquad\qquad\qquad\quad \vdots \\ & c_{m1}x_1 + c_{m2}x_2 + \cdots + c_{mn}x_n \le b_m \\ & x_1, x_2, \ldots, x_n \ge 0 \end{array}$$

which is known as the **standard form**.

Problem

Write the following linear programming problem in standard form

$$\begin{array}{lll} \text{minimise} & -x_1 + 3x_2 - 4x_3 & \\ \text{subject to} & 2x_1 + 4x_3 & \le 4 \\ & x_1 + x_2 - 5x_3 & \ge 4 \\ & 2x_2 - x_3 & \ge 2 \\ & x_1, x_2, x_3 & \ge 0. \end{array}$$

Solution

By a judicious change of signs we can rearrange the problem to

$$\begin{array}{lll} \text{maximise} & x_1 - 3x_2 + 4x_3 & \\ \text{subject to} & 2x_1 + 4x_3 & \le 4 \\ & -x_1 - x_2 + 5x_3 & \le 4 \\ & -2x_2 + x_3 & \le 2 \\ & x_1, x_2, x_3 & \ge 0. \end{array}$$

7.4 Canonical Form

From standard form we move to what proves to be the more useful **canonical form**. We write the non-trivial constraints as equalities. This is achieved by the introduction of extra variables, known as **slack variables**, which are constrained to be ≥ 0. For example, a constraint such as

$$2x_1 + 4x_3 \le 4$$

may be re-written as

$$2x_1 + 4x_3 + x_4 = 4$$
$$x_4 \ge 0$$

by the introduction of the slack variable x_4. The subscript 4 is chosen as being the next available following the sequence of subscripts already allocated.

Problem

Transform the following linear programming problem from standard form to canonical form

$$\begin{array}{ll} \text{maximise} & x_1 - 3x_2 + 4x_3 \\ \text{subject to} & 2x_1 + 4x_3 \leq 4 \\ & -x_1 - x_2 + 5x_3 \leq 4 \\ & -2x_2 + x_3 \leq 2 \\ & x_1, x_2, x_3 \geq 0. \end{array}$$

Solution

Introducing slack variables x_4, x_5 and x_6 (variables x_1, x_2 and x_3 are already in use) we have

$$\begin{array}{ll} \text{maximise} & x_1 - 3x_2 + 4x_3 \\ \text{subject to} & 2x_1 + 4x_3 + x_4 = 4 \\ & -x_1 - x_2 + 5x_3 + x_5 = 4 \\ & -2x_2 + x_3 + x_6 = 2 \\ & x_1, x_2, \ldots, x_6 \geq 0. \end{array}$$

7.5 The Simplex Method

The simplex method is based on an earlier observation, namely that if a problem has an optimal solution then that solution occurs at a vertex of the feasible region (a **feasible vertex**). We assume that the problem is stated in canonical form.

The simplex method chooses a feasible vertex as a starting point and from this determines an edge of the feasible region along which to move to the next feasible vertex. The decision is based on which move maximises the increase in the current value of the objective. Another similar decision is made at the next vertex and the process is repeated until a vertex is reached at which no further increase is possible. If the feasible region is bounded, and this is true of the type of problem in which variables are not allowed to take infinite values, we have a theorem which states that a local optimum is in fact a global optimum. To use a mountaineering analogy, if we go on climbing and reach a point at which we cannot climb any further, then we are assured that the point is the peak of the whole range. The theorem tells us that we are climbing in a mountain range that has only one peak. We use the *garment factory* problem to illustrate how the decision is made at each vertex along the route.

Problem

Write the *garment factory* problem in canonical form. Construct the feasible region using the intersection of trivial and non-trivial constraints. Identify the feasible and non-feasible vertices.

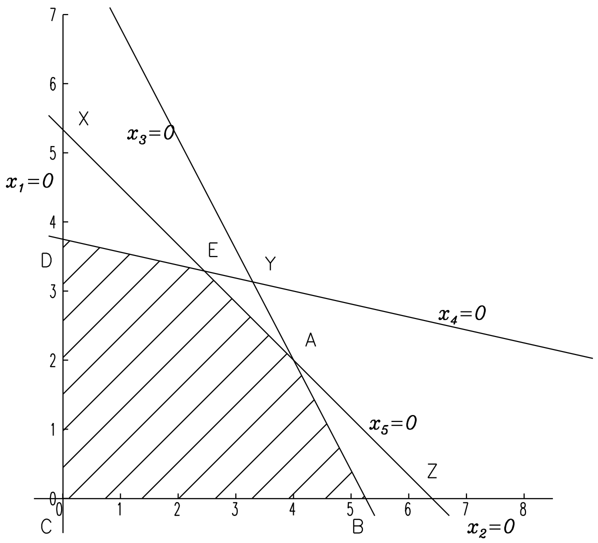


Fig. 7.2 The *garment factory* problem: problem variables and slack variables

Solution

In canonical form the problem becomes

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + 5x_2 + x_3 \quad = 42 \\ & 3x_1 + 16x_2 + x_4 \quad = 60 \\ & 5x_1 + 6x_2 + x_5 \quad = 32 \\ & \qquad x_1, x_2, \ldots, x_5 \geq 0. \end{array}$$

In addition to $x_1 = 0$ and $x_2 = 0$, we use $x_3 = 0$, $x_4 = 0$ and $x_5 = 0$ to identify the feasible region. From Fig. 7.2 it can be seen that the intersection of any pair of lines such as ($x_1 = 0$ and $x_2 = 0$), or ($x_2 = 0$ and $x_4 = 0$), or ($x_1 = 0$ and $x_3 = 0$) determines a point which may or may not be a feasible vertex. In Fig. 7.2 point A, the intersection of $x_3 = 0$ and $x_5 = 0$, is a feasible vertex, whereas point Z, the intersection of $x_2 = 0$ and $x_5 = 0$ is not a feasible vertex. Thus vertices A, B, C, D, E are feasible and vertices X, Y, Z and the intersections of $x_1 = 0$ and $x_3 = 0$, and of $x_2 = 0$ and $x_4 = 0$ are non-feasible.

Discussion

Consideration of Fig. 7.2 shows that each feasible vertex is defined by 2 variables which are zero and 3 which are non-zero. In a more general problem involving n problem variables and m slack variables, each feasible vertex can be defined by setting n of the variables to zero. The n variables in question might be a mixture of problem variables and slack variables. Variables which are set to zero to define a vertex are known as **non-basic variables**. The remaining non-zero variables are

Table 7.2 Feasible vertices of the *garment factory* problem

Vertex	Basic variables	Non-basic variables
A	x_1, x_2, x_4	x_3, x_5
B	x_1, x_4, x_5	x_2, x_3
C	x_3, x_4, x_5	x_1, x_2
D	x_2, x_3, x_5	x_1, x_4
E	x_1, x_2, x_3	x_4, x_5

known as **basic variables**. Furthermore, the set of values of the variables (both problem variables and slack variables) at a feasible vertex is known as a **basic feasible solution**. Table 7.2 shows which variables are basic and which are non-basic, at the feasible vertices A, B, ..., E. It can be seen that moving along an edge of the feasible region from one vertex to the next is equivalent to making a non-basic variable become basic and making a basic variable become non-basic. A feature of the simplex method is the shuffling of variables between the basic and non-basic sets.

It is usual to start with the **all slack solution**, that is the point defined by setting all problem variables to zero. If this point is a feasible vertex then we may proceed. If not we have to look elsewhere for a feasible vertex with which to start the method.

We are now in a position to state the simplex method using terms we have introduced.

- Find a basic feasible solution.
- Consider the possible routes along the edges of the feasible region to nearby vertices. Choose the route which produces the greatest increase in the objective function. If no such route exists we have an optimal solution and the method terminates.
- Move along the chosen route to the next vertex and repeat the previous step.

The following problem indicates how these steps are carried out.

Problem

Use the simplex method to find a solution to the *garment factory* problem.

Solution

For convenience we repeat the canonical form of the problem namely, maximise $z = 500x_1 + 400x_2$ subject to

$$8x_1 + 5x_2 + x_3 = 42 \tag{7.6}$$

$$3x_1 + 16x_2 + x_4 = 60 \tag{7.7}$$

$$5x_1 + 6x_2 + x_5 = 32 \tag{7.8}$$

$$x_1, x_2, \ldots, x_5 \geq 0. \tag{7.9}$$

We start with the all-slack solution, that is $x_1 = 0$ and $x_2 = 0$, a point which from Fig. 7.2 can be seen to be a feasible solution. In the absence of such a diagram we set x_1 and x_2 to zero in (7.6)–(7.8) and find that x_3, x_4 and x_5 take the values 42, 60 and 32 respectively. These values are positive and so none of the constraints of is violated. We therefore have a basic feasible solution.

It is clear that increasing x_1 (from zero) is going to make for a greater increase in the objective z than a corresponding increase in x_2. In short, we look for the non-basic variable with the largest positive coefficient in the objective function. Accordingly we let x_1 become non-zero, that is we make x_1 a basic variable and determine which current basic variable should become non-basic.

Equations (7.6)–(7.8) may be re-written as

$$\begin{aligned} x_3 &= 42 - 8x_1 - \ \ 5x_2 \\ x_4 &= 60 - 3x_1 - 16x_2 \\ x_5 &= 32 - 5x_1 - \ \ 6x_2. \end{aligned}$$

By comparing the ratios 42/8, 60/3 and 32/5 (the constant term divided by the coefficient of x_1) and noting that 42/8 is the smallest of these, it can be seen that as x_1 increases from zero (and x_2 remains zero), the first of x_3, x_4 and x_5 (the non-basic variables) to become zero is x_3. Therefore having let x_1 become basic, x_3 becomes non-basic. We have therefore moved along the edge from C to B in Fig. 7.2.

We repeat the procedure at B. We have to decide which of the non-basic variables becoming non-zero will have the greater effect on the objective z. We have

$$z = 500x_1 + 400x_2$$

which in terms of the non-basic variables (now x_2 and x_3) becomes

$$\begin{aligned} z &= 500(42 - 5x_2 - x_3)/8 + 400x_2 \\ &= 2625 + 87.5x_2 - 62.5x_3. \end{aligned} \tag{7.10}$$

It follows that x_2 should become a basic variable, as x_2 becoming non-zero has greater effect on increasing z than x_3 becoming non-zero. In fact, because the coefficient of x_3 is negative we can exclude it from consideration. To decide which of the basic variables at B should become non-basic we re-write (7.6)–(7.8), with the basic variables gathered on the left-hand side. Since x_3 is to remain non-basic (zero), we have

$$x_1 = (42 - 5x_2)/8 \tag{7.11}$$

$$3x_1 + x_4 = 60 - 16x_2 \tag{7.12}$$

$$5x_1 + x_5 = 32 - 6x_2. \tag{7.13}$$

Using (7.11) in (7.12) and (7.13) we have

$$\begin{aligned} x_4 &= (354 - 113x_2)/8 \\ x_5 &= (46 - 23x_2)/8. \end{aligned} \tag{7.14}$$

Of the ratios 42/5, 354/113 and 46/23, the smallest is 46/23 and so x_5 is the first variable to become zero as x_2 increases from zero. We choose the smallest value so that all the variables remain non-negative. We therefore make x_5 non-basic, which in addition to having made x_2 basic moves us from point B to point A.

We repeat the procedure at A. Once again we look at the objective in terms of the non-basic variables (now x_3 and x_5). Using (7.10) and (7.14) we have

$$\begin{aligned} z &= 2625 - 87.5(8x_5 - 46)/23 - 62.5x_3 \\ &= 2800 - 62.5x_3 - (700/23)x_5. \end{aligned}$$

We are now in the interesting position of being unable to increase the value of the objective since either x_3 or x_5 increasing from zero has a negative effect. We conclude that we are at the optimal solution, namely $z = 2800$. Putting $x_3 = 0$ and $x_5 = 0$ in the constraint equations gives

$$\begin{aligned} 8x_1 + 5x_2 &= 42 \\ 5x_1 + 6x_2 &= 32. \end{aligned}$$

Solving, we find that $x_1 = 4$ and $x_2 = 2$. This is the same solution that we obtained by graphical means.

Discussion

In this particular example we worked our way through the simplex method in a rather ad hoc, manner dealing with problems as they arose. Further problems may arise if one or more of the variables in a basic solution is zero. We may cycle round a sequence of vertices without increasing the value of the objective function.

Since linear programming problems often involve thousands of variables considerable effort has been invested into making computer programs for linear programming as efficient as possible. The most popular version is known as the **revised simplex method** which, with very little extra effort, simultaneously solves a linear programming problem (known as the **dual**) similar to the original in addition to the original (known as the **primal**) and in so doing provides some indication at each stage as to how far we are from the optimal solution and how susceptible the optimal solution is to changes in the data.

7.5.1 Starting the Simplex Method

As already indicated, if the all-slack solution is feasible then this is an obvious choice with which to start the simplex method. If the all-slack solution is not feasible we must look elsewhere. Though possible for smaller problems, randomly selecting variables to be non-basic (zero) could be too time consuming if applied to larger

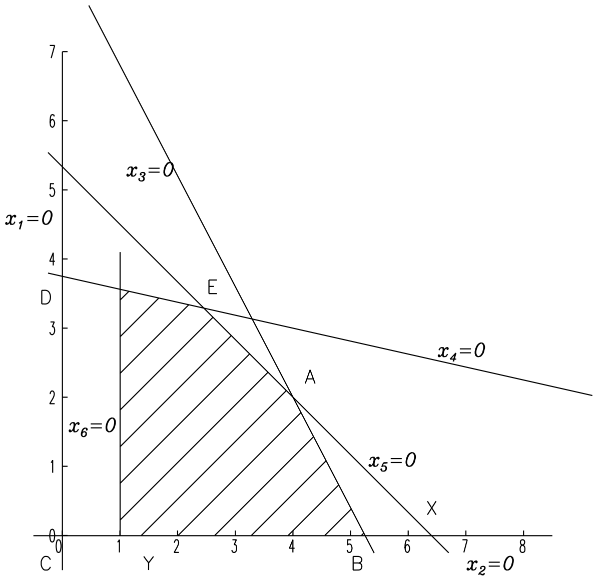


Fig. 7.3 Feasible region for the revised *garment factory* problem

problems. The next problem, a variation on the *garment factory* problem, illustrates a more sophisticated technique. By choice of a suitable objective we let the simplex method itself find a basic feasible solution.

Problem

In addition to existing constraints the garment factory is required to produce at least one batch of all-weather raincoats per day. Find a basic feasible solution to this problem.

Solution

Using the previous notation we add the constraint $x_1 \geq 1$ and so the problem has the canonical form

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + \ 5x_2 \quad +x_3 \quad = 42 \\ & 3x_1 + 16x_2 \quad +x_4 \quad = 60 \\ & 5x_1 + \ 6x_2 \quad +x_5 \quad = 32 \\ & x_1 \qquad\qquad -x_6 \quad = 1 \\ & \qquad x_1, x_2, \ldots, x_6 \geq 0. \end{array}$$

The feasible region is shown in Fig. 7.3. The all-slack solution found by setting the problem variables (x_1 and x_2) to zero no longer gives a feasible vertex. Not only can it be seen from Fig. 7.3 that point C is outside the feasible region, but also $x_1 = 0$ implies $x_6 = -1$, which breaks the constraint $x_6 \geq 0$.

Undaunted we start with the all-slack solution but temporarily replace the original objective by what is known as a **pseudo objective** z'. The pseudo objective is chosen with the aim of increasing the value of x_6 and hopefully making it positive. We use the simplex method with the objective z' given by $z' = x_6$ and the original constraints.

We have $z' = x_6 = x_1 - 1$ and so choosing x_1 rather than x_2 to become non-zero from the list of non-basic variables (x_1, x_2) has the greater effect on z'. So x_1 becomes a basic variable. The basic variables are currently x_3, x_4, x_5 and x_6, so keeping x_2 zero we have

$$\begin{aligned} x_3 &= 42 - 8x_1 \\ x_4 &= 60 - 3x_1 \\ x_5 &= 32 - 5x_1 \\ x_6 &= -1 + x_1. \end{aligned}$$

As x_1 increases from zero it is clear that x_6 will be the first variable to become zero, and so x_6 becomes non-basic. We now have $x_2 = 0$ and $x_6 = 0$, from which it follows that $x_1 = 1$, $x_3 = 34$, $x_4 = 57$ and $x_5 = 27$. This is a basic feasible solution to the original problem and so it can be used to start the simplex method proper.

Discussion

In effect one step of the simplex method using the pseudo objective has taken us from point C to point Y in Fig. 7.3. If this had not been a feasible vertex we would have tried another step but with a different pseudo objective; in such a case the pseudo objective would be the sum of all the negative variables. If the original linear programming problem involves equality constraints the following problem shows how to overcome this particular difficulty.

Problem

In addition to the original constraints the garment factory is required to produce exactly 6 batches of coats per day. The requirement that at least one batch of all-weather raincoats be produced no longer applies. Find a basic feasible solution.

Solution

The problem has the canonical form

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + 5x_2 + x_3 = 42 \\ & 3x_1 + 16x_2 + x_4 = 60 \\ & 5x_1 + 6x_2 + x_5 = 32 \\ & x_1 + x_2 + x_6 = 6 \\ & x_1, x_2, \ldots, x_5 \geq 0, \quad x_6 = 0. \end{array}$$

In addition to the usual slack variables we introduce a so-called **artificial variable** x_6. The variable is artificial in the sense that, unlike the problem variables and the slack variables, it is required to be zero. As with the previous problem an all-slack solution is not possible since setting the problem variables (x_1 and x_2) to zero implies $x_6 = 6$ which breaks the constraint $x_6 = 0$.

Since we wish to reduce the value of x_6 (to zero) we use one step of the simplex method with the objective z' given by $z' = -x_6$ and the original constraints. Thus we are seeking to maximise $-x_6$, which is to minimise x_6.

We have

$$\begin{aligned} z' &= -x_6 \\ &= x_1 + x_2 - 6 \end{aligned}$$

and so we choose x_1 to become a basic variable. In this case we could have equally well chosen x_2. The basic variables are currently x_3, x_4, x_5 and x_6, so keeping x_2 zero we have

$$\begin{aligned} x_3 &= 42 - 8x_1 \\ x_4 &= 60 - 3x_1 \\ x_5 &= 32 - 5x_1 \\ x_6 &= 6 - x_1. \end{aligned}$$

As x_1 increases from zero it is clear that x_3 will be the first variable to become zero, and so x_3 becomes non-basic. We now have $x_2 = 0$ and $x_3 = 0$, from which it follows that $x_1 = 6$, $x_4 = 42$, $x_5 = 2$ and $x_6 = 0$. None of the constraints are violated and so we have a basic feasible solution with which to start the simplex method.

Discussion

In general, if the all-slack solution cannot be used to start the simplex method because it is not a basic feasible solution, we look for a basic feasible solution using the pseudo objective

$$\sum_{x_i<0} x_i - \sum_{\substack{x_i>0 \\ x_i \text{ artificial}}} x_i.$$

7.6 Integer Programming

We reconsider the original *garment factory* problem but simplified to the extent that just two processes are involved, namely cutting and sewing. However we add the constraint that only complete batches of raincoats can be manufactured in any one day. Once again we look for the most profitable manufacturing strategy.

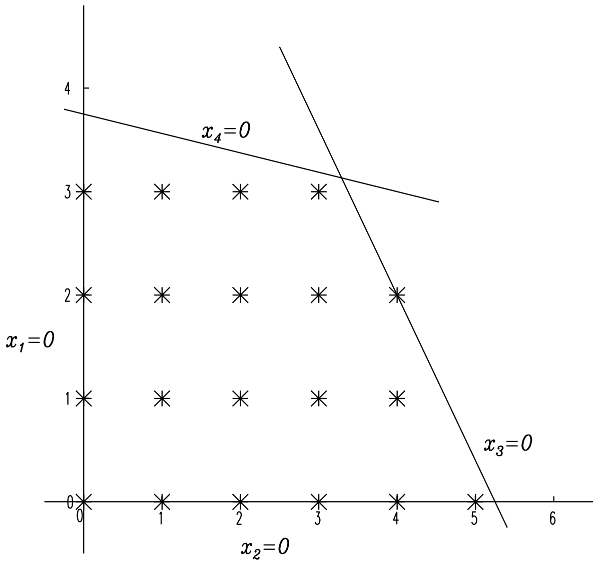


Fig. 7.4 Feasible region for the *integer garment* problem

The canonical form of this latest version of the problem, which we refer to as the *integer garment* problem, is

$$\begin{aligned} \text{maximise} \quad & z = 500x_1 + 400x_2 \\ \text{subject to} \quad & 8x_1 + \;\, 5x_2 + x_3 \;\; = 42 \\ & 3x_1 + 16x_2 + x_4 = 60 \\ x_1, x_2, & x_3, x_4 \geq 0 \text{ and integer.} \end{aligned}$$

The problem as stated is an example of an **integer programming** problem. If x_1 and x_2 were not required to be integer, the optimal solution would be given by the intersection of

$$\begin{aligned} 8x_1 + \;\, 5x_2 &= 42 \\ \text{and} \quad 3x_1 + 16x_2 &= 60 \end{aligned}$$

namely, $x_1 = 3.3$ and $x_2 = 3.1$, values which may be either calculated directly or read from Fig. 7.4. Since x_1 and x_2 are required to be integer it might seem appropriate to take the nearest whole number values as the solution, namely, $x_1 = 3$ and $x_2 = 3$ which gives a z value of 2700. However this is not the optimal solution. Setting $x_1 = 4$ and $x_2 = 2$, which is acceptable since this keeps x_3 and x_4 positive, gives z a higher value of 2800. Clearly an alternative strategy is required.

The feasible region for the *integer garment* problem is shown in Fig. 7.4 and consists of a number of discrete points. This type of discrete region is in contrast to the feasible regions of general problems, which we refer to as **continuous problems**, in which variables may take both integer and non-integer values. We solve integer programming problems by the **branch and bound** method. By solving a number of continuous problems the method aims to reduce the feasible region to that of sub-regions containing a feasible solution, one or more of which will be an optimal solution. It should be noted that a solution to an integer programming problem is not necessarily unique.

7.6.1 The Branch and Bound Method

Given an integer programming problem, we begin by solving the corresponding continuous problem by the simplex method. If we obtain an all-integer solution then we have solved the problem. If, however, one or more of the variables of the optimal solution does not have an integer value then we choose one such variable on which to **branch**. That is we form two further problems with a view to splitting the feasible region into two sub-regions. For example, in the *integer garment* problem, having solved the initial continuous problem and found that x_1 is not integer at the solution, we could decide to branch on x_1. In this case we have a solution $x_1 = 3.3$, so to progress towards an all-integer solution we seek to reduce the feasible region by solving two further (continuous) problems, the original continuous problem with the added constraint (an added **bound**) $x_1 \leq 3$ and the original continuous problem with the added constraint $x_1 \geq 4$. Hence the name **branch and bound**. The process continues until no further progress can be made along a particular branch or because it becomes apparent that further progress would not offer any improvement on existing solutions. The following example illustrates the method.

Problem

Solve the reduced *integer garment* problem by the branch and bound method, that is

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + 5x_2 \leq 42 \\ & 3x_1 + 16x_2 \leq 60 \\ \multicolumn{2}{c}{x_1, x_2 \geq 0, \quad x_1, x_2 \quad \text{integer.}} \end{array}$$

Solution

The solution to the associated continuous problem, which we refer to as Problem (1) is, as we have already seen, $x_1 = 3.3$, $x_2 = 3.1$. We branch on x_1 and form the following two sub-problems:

Problem (1.1)

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + 5x_2 \leq 42 \\ & 3x_1 + 16x_2 \leq 60 \\ & x_1 \leq 3 \\ \multicolumn{2}{c}{x_1, x_2 \geq 0, \quad x_1, x_2 \quad \text{integer}} \end{array}$$

Problem (1.2)

$$\begin{array}{ll} \text{maximise} & z = 500x_1 + 400x_2 \\ \text{subject to} & 8x_1 + 5x_2 \leq 42 \\ & 3x_1 + 16x_2 \leq 60 \\ & x_1 \geq 4 \\ \multicolumn{2}{c}{x_1, x_2 \geq 0, \quad x_1, x_2 \quad \text{integer.}} \end{array}$$

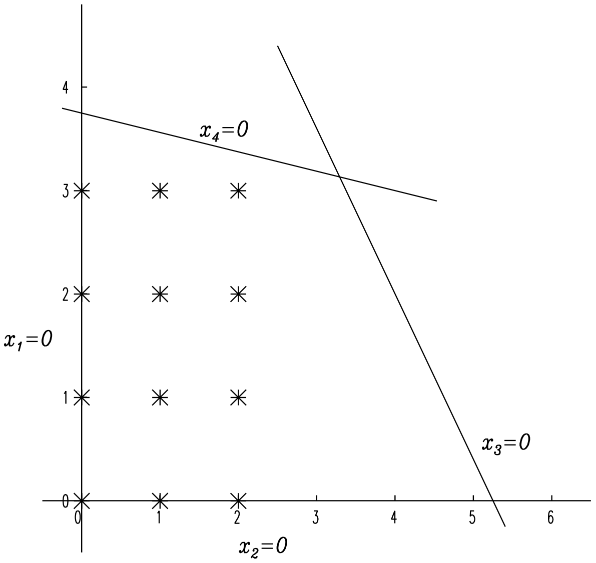


Fig. 7.5 Feasible region for Problem (1.1)

The feasible region for Problem (1.1) is shown in Fig. 7.5. Solving by the simplex method gives the solution $x_1 = 3$, $x_2 = 3.19$. This is not an integer solution so we branch on the offending variable x_2. This gives rise to two further sub-problems:

Problem (1.1.1)		Problem (1.1.2)	
maximise	$z = 500x_1 + 400x_2$	maximise	$z = 500x_1 + 400x_2$
subject to	$8x_1 + \ \ 5x_2 \le 42$	subject to	$8x_1 + \ \ 5x_2 \le 42$
	$3x_1 + 16x_2 \le 60$		$3x_1 + 16x_2 \le 60$
	$x_1 \le 3$		$x_1 \le 3$
	$x_2 \le 3$		$x_2 \ge 4$
$x_1, x_2 \ge 0,$	x_1, x_2 integer	$x_1, x_2 \ge 0,$	x_1, x_2 integer.

Solving Problem (1.1.1) by the simplex method gives the solution $x_1 = x_2 = 3$ and a value of 2700 for the objective. This is an integer solution and so is a contender for the solution to the integer *integer garment* problem. There is nothing to be gained in subdividing Problem (1.1.1) any further since this would be to look for integer solutions in a feasible region fully contained within the feasible region for Problem (1.1.1). Such solutions could only give a value of the objective less than that already obtained. Accordingly we look at problem (1.1.2). This is an **infeasible problem**, that is a problem without a solution, since $x_2 \ge 4$ and the constraint $3x_1 + 16x_2 \le 60$ imply $x_1 < 0$, which is not allowed.

Having exhausted the possibilities on the left-hand side of the tree of Fig. 7.6, we turn our attention to the right-hand side and Problem (1.2). This has the solution $x_1 = 4$, $x_2 = 2$ and a value of 2800 for the objective. For reasons already stated in connection with Problem (1.1.1), there is no point in going any further. Of the (two) integer solutions we have found Problem (1.2) gives the higher value for the objective and so a solution to the integer *integer garment* problem has been found.

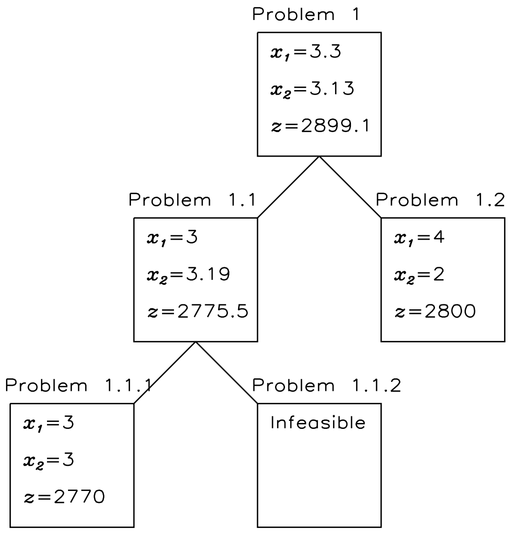


Fig. 7.6 *Integer garment* problem, branch and bound method

Discussion

The branch and bound method does not offer a unique route to a solution. For example, we could have formed two new problems by branching on x_2 rather than x_1. In this case we would have produced a different *tree* but would have ultimately found a solution though not necessarily the same as found by branching first on x_1. If we had initially branched on x_2, there would have been no need to progress beyond what we have labelled Problem (1.1) in Fig. 7.6, since at this point the objective is less than 2800. In general the advice is to branch on the variable having greatest influence on the objective as this is found to offer a quicker route to a solution.

7.7 Decision Problems

We consider a different kind of business model for the garment factory, which is deciding whether to concentrate production on just one type of raincoat. If as before x_1 represents the number of batches of all-weather raincoats and x_2 the number of luxury-lined, we add the following constraints to earlier linear programming models

$$\begin{aligned} x_1 &\le 1000d \\ x_2 &\le 1000(1-d) \\ d &\le 1 \\ d &\ge 0, \quad d \text{ integer}. \end{aligned} \tag{7.15}$$

Table 7.3 Car hire company, model data

	Auto. gears	Sun roof	Child seat	Hatch-back	CD	Route finder	Immobiliser
Model 1	Yes	No	Yes	No	Yes	No	No
Model 2	Yes	No	No	Yes	No	Yes	No
Model 3	No	Yes	Yes	No	No	Yes	No
Model 4	No	No	No	Yes	No	Yes	Yes
Model 5	Yes	Yes	No	No	No	No	Yes
Model 6	Yes	No	No	No	Yes	Yes	No

Here d is a **decision variable**. A decision variable is a problem variable constrained to take either the value 0 or 1. The number 1000 is somewhat arbitrary; any number as large or larger than the greater of the possible x_1 and x_2 values will do. If d takes the value 1 in the optimal solution this indicates that all-weather raincoats are to be made, if d takes the value 0 this indicates luxury-lined are to be made. Examples of the use of decision variables follow.

Problem

A car hire company offers a number of models to its customers. As can be seen from Table 7.3, not all models have the same features. The car hire company wishes to reduce the number of models available to the minimum while retaining sufficient models to offer at least one car having at least one of the current seven features on offer. Formulate the problem the company faces as a linear programming problem.

Solution

We use decision variables $d_1, d_2, \ldots, d_6$. $d_i = 1$ indicates that model i is to be retained; $d_i = 0$ indicates that model i is not to be retained. The objective z which we seek to minimise is the number of models, therefore

$$z = d_1 + d_2 + d_3 + d_4 + d_5 + d_6.$$

The requirement that at least one model is available with automatic gears may be expressed as

$$d_1 + d_2 + d_5 + d_6 \geq 1$$

since only models $1, 2, 5$ and 6 have this feature. Similar expressions apply to the other features, so we have

$$\begin{array}{lrl}
\text{minimise} & z = d_1 + d_2 + d_3 + d_4 + d_5 + d_6 & \\
\text{subject to} & d_1 + d_2 + d_5 + d_6 \geq 1 & \text{Auto. gears} \\
 & d_3 + d_5 \geq 1 & \text{Sun roof} \\
 & d_1 + d_3 \geq 1 & \text{Child seat} \\
 & d_2 + d_4 \geq 1 & \text{Hatch-back} \\
 & d_1 + d_6 \geq 1 & \text{CD} \\
 & d_2 + d_3 + d_4 + d_6 \geq 1 & \text{Route finder} \\
 & d_4 + d_5 \geq 1 & \text{Immobiliser} \\
 & d_1, d_2, \ldots, d_6 \geq 0 & \\
 & d_1, d_2, \ldots, d_6 \leq 1 & \\
 & d_1, d_2, \ldots, d_6 \quad \text{integer.} &
\end{array}$$

A computer solution to this integer programming problem is discussed in Exercise 8.

7.8 The Travelling Salesman Problem

As a further example of how the skill in successful linear programming lies in the modelling of the problem rather than applying the standard method of solution, we consider one of the most famous of all problems, known as **the travelling salesman problem**. The problem was originally stated in terms of a salesman having to visit a number of cities and wishing to devise a route to minimise travel costs. We consider a particular example. A van driver is based in Manchester with deliveries to make to depots in Leeds, Stoke, Liverpool and Preston. The van driver wishes to devise a round trip which is the shortest possible. The relevant distances are shown in Table 7.4.

To simplify matters we identify the five locations Manchester, Leeds, Stoke, Liverpool and Preston by the numbers $1, 2, \ldots, 5$ respectively. We introduce decision variables d_{ij} to indicate whether or not on leaving i the next stop is j. For example we take $d_{12} = 1$ to mean that the driver travels directly from Manchester to Leeds, on the other hand $d_{12} = 0$ means that the route from Manchester to Leeds is not direct. Similarly, $d_{42} = 1$ implies that the route from Liverpool to Leeds is direct, but is indirect if $d_{42} = 0$. Note that d_{ij} has a different meaning from d_{ji}.

Table 7.4 Distances (miles)

	Leeds	Stoke	Liverpool	Preston
Manchester	44	45	35	32
Leeds		93	72	68
Stoke			57	66
Liverpool				36

The objective z which we are seeking to minimise is given by

$$z = \sum_{\substack{i,j=1 \\ i \neq j}}^{5} m_{ij} d_{ij}$$

where m_{ij} is the distance between depots i and j, as shown in the table above.

Now for the constraints. The solution is to take just one route out of Manchester and this may be expressed as

$$d_{12} + d_{13} + d_{14} + d_{15} = 1.$$

Similarly, having made the first call, wherever that happens to be, we require just one route out. This applies to each of the other locations and so we also have

$$\begin{aligned} d_{21} + d_{23} + d_{24} + d_{25} &= 1 \\ d_{31} + d_{32} + d_{34} + d_{35} &= 1 \\ d_{41} + d_{42} + d_{43} + d_{45} &= 1 \\ d_{51} + d_{52} + d_{53} + d_{54} &= 1. \end{aligned}$$

So far so good, but this is not a complete model. We do not allow solutions which include visiting the same depot twice on the grounds that a circular route is bound to be more direct. In order to exclude combinations such as Manchester to Liverpool ($d_{14} = 1$) and Preston to Liverpool ($d_{54} = 1$) we need to specify that in addition to just one route leading out, each location has just one route leading in. We have

$$\begin{aligned} d_{21} + d_{31} + d_{41} + d_{51} &= 1 \\ d_{12} + d_{32} + d_{42} + d_{52} &= 1 \\ d_{13} + d_{23} + d_{43} + d_{53} &= 1 \\ d_{14} + d_{24} + d_{34} + d_{54} &= 1 \\ d_{15} + d_{25} + d_{35} + d_{45} &= 1. \end{aligned}$$

In addition we have

$$d_{ij} \geq 0, \qquad d_{ij} \leq 1, \qquad d_{ij} \quad \text{integer.}$$

A computer solution is discussed in Exercise 9.

7.9 The Machine Scheduling Problem

As a final example of forming a linear programming model we consider a **machine scheduling problem**. In general this is concerned with allocating machines, or processes, as part of a larger process in such a way that the total work in hand may be completed as quickly as possible. To illustrate the technique we consider the following problem.

Table 7.5 Paint shop process times

	Car	Van
Clean	13	11
Spray	10	22
Dry	6	9

Problem

A paint shop uses three processes, namely cleaning, spraying and drying (in that order), to re-paint cars and vans. The facilities of the paint shop are limited in that only one vehicle can be cleaned at any one time, and a similar restriction applies to spraying and drying. The time taken (in minutes) for each process is shown in Table 7.5. The paint shop has an order to paint 2 cars and a van and wishes to organise the work so that it can be completed in the shortest time possible. Formulate the decision as to how to organise the work as a linear programming problem.

Solution

To simplify matters we refer to the two cars as items 1 and 2 and the van as item 3. The processes of cleaning, spraying and drying are referred to as processes 1, 2 and 3 respectively. We assume that time is measured from zero and the time at which item i starts process j is given by t_{ij}.

Since spraying follows cleaning, we have

$$t_{12} \geq t_{11} + 13$$
$$t_{22} \geq t_{21} + 13$$
$$t_{32} \geq t_{31} + 11$$

and since drying follows spraying, we have

$$t_{13} \geq t_{12} + 10$$
$$t_{23} \geq t_{22} + 10$$
$$t_{33} \geq t_{32} + 22.$$

We assume that car 1 (item 1) takes precedence over car 2 (item 2). Since both items are essentially the same, this will not affect the final outcome. Therefore, we have

$$t_{11} + 13 \leq t_{21}$$
$$t_{12} + 10 \leq t_{22}$$
$$t_{13} + 6 \leq t_{23}.$$

To model the decision as to the order in which vehicles are to be processed we introduce decision variables d_{ijk}. We take $d_{ijk} = 1$ to mean that for process i, item j is processed before item k and that $d_{ijk} = 0$ means otherwise.

If, for example, item 1 is to go through process 1 before item 3, that is if the first car is to be cleaned before the van, we have

$$t_{11} + 13 \le t_{31} \quad \text{and} \quad d_{113} = 1, \qquad d_{131} = 0.$$

If this were not the case, we would have

$$t_{31} + 11 \le t_{11} \quad \text{and} \quad d_{131} = 1, \qquad d_{113} = 0.$$

These two constraints may be written as

$$\begin{aligned} t_{11} + 13 &\le t_{31} + 1000(1 - d_{113}) \\ t_{31} + 11 &\le t_{11} + 1000 d_{113}. \end{aligned}$$

This device neatly expresses the either–or condition in a manner similar to that shown in (7.15). If, for example, $d_{113} = 1$, the second of the constraints is automatically satisfied no matter what the values of t_{31} and t_{11} (within reason) and so has no impact. The constraint has been placed well outside the feasible region. Similarly, if $d_{113} = 0$ the first constraint becomes redundant and the second takes effect. Note that there is now no need for a separate variable d_{131}.

We can write similar expressions by regarding items 2 and 3 for process 1:

$$\begin{aligned} t_{21} + 13 &\le t_{31} + 1000(1 - d_{123}) \\ t_{31} + 11 &\le t_{21} + 1000 d_{123}. \end{aligned}$$

Similar expressions apply to the other processes. For process 2

$$\begin{aligned} t_{12} + 10 &\le t_{32} + 1000(1 - d_{213}) \\ t_{32} + 22 &\le t_{12} + 1000 d_{213} \\ t_{22} + 10 &\le t_{32} + 1000(1 - d_{223}) \\ t_{32} + 22 &\le t_{22} + 1000 d_{223} \end{aligned}$$

and for process 3

$$\begin{aligned} t_{13} + 6 &\le t_{33} + 1000(1 - d_{313}) \\ t_{33} + 9 &\le t_{13} + 1000 d_{313} \\ t_{23} + 6 &\le t_{33} + 1000(1 - d_{323}) \\ t_{33} + 9 &\le t_{23} + 1000 d_{323}. \end{aligned}$$

To complete the specification of the constraints we have for relevant i and j

$$\begin{gathered} t_{ij} \ge 0 \\ d_{i13}, d_{i23} \ge 0, \qquad d_{i13}, d_{i23} \le 1 \\ d_{i13}, d_{i23} \quad \text{integer.} \end{gathered}$$

We now consider the objective z. We are looking to complete all three processes for all three items in as short a time as possible, so we are looking to minimise whatever happens to be the greater value of $t_{23} + 6$ and $t_{33} + 9$, the times at which

Table 7.6 Paint shop optimum schedule

	Starting times		
	car 1	car 2	van
clean	17	30	0
spray	33	43	11
dry	43	53	33

the second car and the van have completed the final process. We know that by this time the first car will have been through the system. To avoid having to specify z directly we state the problem as

$$\begin{aligned} &\text{minimise} && z \\ &\text{subject to} && z \geq t_{23} + 6 \\ & && z \geq t_{33} + 9 \end{aligned}$$

and constraints already stated.

Discussion

Solving by the simplex method with branch and bound gives a value of 59 for the objective. There are a number of possible t_{ij} values, a particular solution is shown in Table 7.6. The van is the first vehicle to be cleaned, followed by car 1 and then car 2. For each process the order of the vehicles is the same. This particular solution leaves the cleaning process idle from time $t = 11$ to time $t = 17$. Whilst this does not affect the eventual outcome it may not be the solution we would wish to implement. We might prefer to operate the cleaning process continually until time $t = 36$. In general such requirements would be modelled by means of additional constraints.

Summary In this chapter we have shown how linear programming can be used to solve problems which might arise in the business and economic world. For this reason the term **Operations Research** is often used to describe the application of linear programming techniques. We saw how a simple graphical approach could be applied to a problem involving two variables and used this as a basis for forming a more powerful algebraic method which culminated in the simplex method. To achieve this we defined a canonical form for linear programming problems and showed how problems could be presented in this form, if necessary by introducing slack variables. We showed how the simplex method could be regarded as a progression from feasible vertex to feasible vertex and that the all-slack solution, provided it is feasible, is a convenient starting point. In the event of this not being the case, we showed how a feasible vertex could be found by means of a pseudo objective.

We identified the need for integer solutions and this led to the branch and bound method, which effectively uses the simplex method repeatedly until the feasible region is reduced to the point at which further investigation is unnecessary.

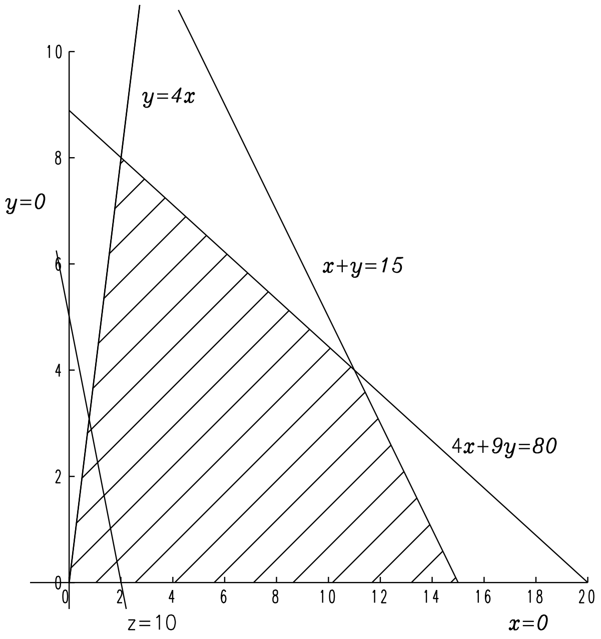


Fig. 7.7 Feasible region for Exercise 1

Finally we showed how linear programming can model the decision-making process through the use of decision variables. This is an area which offers scope for considerable ingenuity in the modelling process and as such may present more problems than simply applying the solution process. In this context we examined the travelling salesman problem and the machine scheduling problem.

Exercises

1. Sketch or use Matlab to produce a diagram to confirm Fig. 7.7 is the feasible region for the following linear programming problem.

$$\begin{array}{lr} \text{maximise} & z = 5x + 2y \\ \text{subject to} & x + y \leq 15 \\ & 4x + 9y \leq 80 \\ & 4x - y \geq 0 \\ \text{and} & x, y \geq 0. \end{array}$$

A Matlab program based on the following scheme would be appropriate

```
% draw the line y = 4x using 100 equally spaced x values in the range
x = linspace( 0, 20, 100 );
pc set the axis limits
axis ([ 0 20 0 10 ]);
y = 4*x;
```

```
plot( x, y );
% hold the plot for the next line, y = 15 - x
hold;
y = 15−x;
% use the same x-axis range
axis ([0 20 0 20]);
plot( x, y );
% plot remaining lines similarly
...
...
% use a different colour (red) for z = 10
plot( x, y, 'r' );
% annotate the graph using the cross-hairs to position the text
gtext('feasible region' )
gtext('z = 10' )
% axes labels, titles and other annotations may be added interactively
% using the drop-down menu under the Insert option on the resulting
% Figure menu
```

As a line parallel to $z = 10$ moves across the feasible region it can be seen that the maximum value of z will be reached at the intersection of $x + y = 15$ and $y = 0$. The solution is therefore $x = 15$, $y = 0$, at which point $z = 75$.

The solution may also be obtained using the Matlab function *linprog* with a parameter list (f, A, b) which finds the solution to the linear programming problem

$$\text{minimise} \quad f(\mathbf{x}) \qquad \text{subject to} \quad \mathbf{Ax} \le b.$$

A sequence of commands appropriate to the current example would be

```
f = [ −5  −2]
A = [ 1 1; 4 9; −4 1; −1 0; 0 −1]
b = [ 15 80 0 0 0]
x = linprog( f, A, b );
```

Note that the problem has to be re-stated as a minimisation problem in standard form. In particular the requirements $x >= 0$ and $y >= 0$ become $-x <= 0$ and $-y <= 0$ respectively.

2. Consider the following linear programming problems, all of which are unusual in some way. Sketch or use Matlab to illustrate each problem. Identify cases for which the feasible region is not bounded or does not exist and any redundant constraints. Find the solutions, if they exist either directly from the plots or by using *linprog*. If you intend to produce one or more of the examples in the same session use the command *figure* to start a new figure while retaining an existing figure.

$$\begin{array}{rl} \text{maximise} & z = x + y \\ \text{subject to} & 5x + y \geq 20 \\ & 5x + 2y \geq 40 \\ & 18x - 5y \leq 90 \\ \text{and} & x, y \geq 0. \end{array} \qquad \begin{array}{rl} \text{minimise} & z = 5y - x \\ \text{subject to} & x - 3y \geq 1 \\ & x + 2y \geq 3 \\ & 2x + y \geq 4 \\ \text{and} & x, y \geq 0. \end{array}$$

$$\begin{array}{rl} \text{minimise} & z = 3x - 2y \\ \text{subject to} & x - 2y \geq 2 \\ & 2y - 5x \geq 10 \\ \text{and} & x, y \geq 0. \end{array} \qquad \begin{array}{rl} \text{maximise} & z = 10x + 3y \\ \text{subject to} & y - x \leq 1 \\ & 3x + y \geq 3 \\ & 3y - x \leq 2 \\ & 3x + 4y \leq 12 \\ \text{and} & x, y \geq 0. \end{array}$$

3. A factory produces two types of bed, the Sleep-eezy and the Ortho. The beds are made of fabric, wood and metal. The amount of each material (in appropriate units) which is available for the daily production run is shown in Table 7.7. Table 7.7 also shows the quantity of material required for each model and the retail value of each bed. The factory can sell everything that it produces. Assuming that the aim is to maximise the value of retail sales. Denoting the number of Sleep-eezy beds to be made by x and the number of Ortho beds by y, the objective function z, the total retail value, is given by

$$z = 200x + 400y.$$

The constraints arise from the material available. Taking each in turn we have

$$5x + 12y \leq 120 \quad \text{(fabric)} \tag{7.16}$$

$$10x + 7y \leq 140 \quad \text{(metal)} \tag{7.17}$$

$$12x + 15y \leq 180 \quad \text{(wood)} \tag{7.18}$$

and in addition $x, y \geq 0$. Sketch or use Matlab to produce a diagram to confirm Fig. 7.8 is the feasible region for the following linear programming problem. Use either the diagram or Matlab to decide how many beds of each type should be made each day in order to maximise retail value. As a line parallel to $z = 1600$ moves across the feasible region it can be seen that the maximum value of z will be reached at the intersection of $12x + 15y = 180$ and $5x + 12y = 120$. The solution by inspection, or solving the equations, is found to be $x = 5.2$, $y = 7.8$, at which point $z = 4173.9$. Whether such a fractional solution is helpful to the company is another matter (see below).

Table 7.7 Exercise 3, data

	Availability	Sleep-eezy	Ortho
Fabric	120	5	12
Metal	140	10	7
Wood	180	12	15
Retail value (£)		200	400

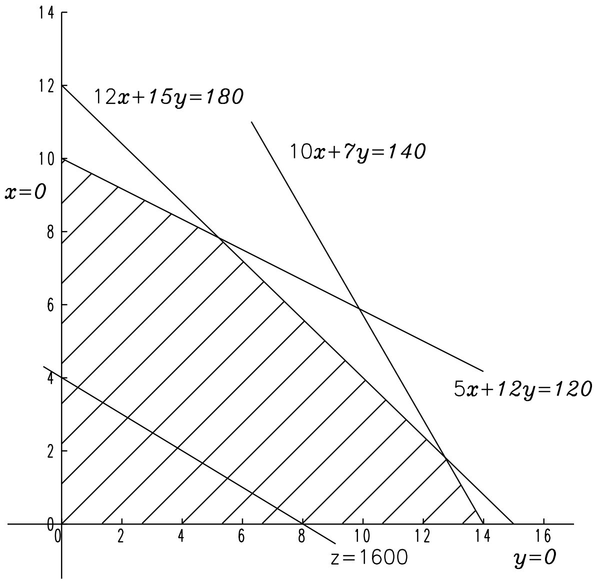


Fig. 7.8 Feasible region for Exercise 3

4. Reconsider the previous exercise but with the added restriction that once started a bed must be completed within the daily production run. The problem is now an integer programming problem.

   The solution to the continuous problem was found to be $x = 5.2$, $y = 7.8$. Branching on y since this variable has the greater effect on the objective produces two sub-problems, namely

$$\begin{array}{rl}
\text{maximise} & z = 200x + 400y \\
\text{subject to} & 5x + 12y \le 120 \\
& 10x + 7y \le 140 \\
& 12x + 15y \le 180 \\
& y \le 7 \\
\text{and} & x, y \ge 0.
\end{array}
\qquad
\begin{array}{rl}
\text{maximise} & z = 200x + 400y \\
\text{subject to} & 5x + 12y \le 120 \\
& 10x + 7y \le 140 \\
& 12x + 15y \le 180 \\
& y \ge 8 \\
\text{and} & x, y \ge 0.
\end{array}$$

   Solve these problems using the Matlab function *linprog* using a command of the form $[sol\ val] = linprog(f, A, b)$. On exit vector *sol* will hold the values of the problem variables, *val* will hold the value of the objective.

   Verify that the solutions to the problems are ($x = 6.25$, $y = 7$) and ($x = 4.8$, $y = 8$) with objectives 4050 and 4160 (allowing for changes of sign because of having to state the problem as a minimisation).

   Branch on x in the first of these solutions, and consider the further two sub-problems ($x \le 6, y \le 7$) and ($x \ge 7, y \le 7$). Show the first produces an all-integer solution ($x = 6$, $y = 7$) with objectives 4000 and that the second produces a solution with a smaller objective. There is therefore no need to proceed further down this particular route, the current objective (4000) cannot be equalled or exceeded.

Table 7.8 Exercise 4, branch and bound progress

Solutions			Extra constraints
x	y	z	
5.2	7.8	4174	
6.25	7	4050	$y \leq 7$
6	7	4000	$x \leq 6$, $y \leq 7$
(z will be ≤ 4000 on this branch)			
7	6.4	3960	$x \geq 7$, $y \leq 7$
(z will be ≤ 3960 on this branch)			
4.8	8	4160	$y \geq 8$
4	8	4000	$x \leq 4$, $y \geq 8$
(z will be ≤ 4000 on this branch)			
no solution			$x \geq 5$, $y \geq 8$
(and no more solutions on this branch)			

Proceed in similar manner down the $y \geq 8$ route until further integer solutions are found, or further progress is either unnecessary or not possible. Obtain the results shown in Table 7.8, which confirm that the objective 4000 is an optimum solution to the problem and that in addition to the solution for $x = 6$, $y = 7$ there is also solution $x = 4$, $y = 8$ producing the same optimum. Note that the table does not exclude the possibility of further solutions. An initial branch on y rather than x would have found a further solution $x = 2$, $y = 9$.

5. Matlab provides the function *bintprog* for solving the binary integer programming problem

$$\text{minimise} \quad f(\mathbf{x}) \qquad \text{subject to} \quad \mathbf{Ax} \leq b$$

where the elements of X are binary integers, i.e., 0's or 1's. The relevant command is $X = bintprog(f, A, b)$.

Use *bintprog* to find a solution to the previous question, writing variables x and y as strings of four binary digits (four being quite sufficient since both variables x and $y < 15$). Use the following Matlab program

```
u = [ 1  2  4  8 ] % Binary units
A = [5∗u   12∗u ; 10∗u   7∗u; 12∗u   15∗u ] % Converting binary x and y
                                              % to decimal
f = [−200*u −400*u] % Re-state as a minimisation problem
b = [ 120: 140; 180]
sol = bintprog( f , A , b)
```

to show that on completion *sol* will hold a binary string corresponding to two decimal numbers 4 and 8, which was one of the solutions found in the previous question.

6. The all-slack solution to the following linear programming problem is not feasible, since setting the problem variables contravenes the constraints.

$$\begin{array}{ll} \text{minimise} & z = 3x_2 - x_1 - 4x_3 \\ \text{subject to} & 2x_1 + 4x_3 \leq 4 \\ & x_1 + x_2 + x_3 = 3 \\ & 2x_2 - x_3 \geq 2 \\ \text{and} & x_1, x_2, x_3 \geq 0. \end{array}$$

However by writing the problem in canonical form and introducing slack variables x_4, x_6 and an artificial variable x_5 a basic feasible solution may be found with which to start the simplex method. We have

$$2x_1 + 4x_3 + x_4 = 4$$
$$x_1 + x_2 + x_3 + x_5 = 3$$
$$-2x_2 + x_3 + x_6 = -2$$
$$x_5 = 0$$
$$x_1, x_2, x_3, x_4, x_6 \geq 0.$$

Setting the problem variables (x_1, x_2, x_3) to zero produces $x_4 = 4$, $x_5 = 3$ and $x_6 = -2$ which contravenes the constraints. In order to reduce x_5 (to zero) and increase x_6 (to a non-negative value) minimise the pseudo objective $z = x_5 - x_6$. The Matlab function *linprog* has an extended form X = linprog(f, A, b, Aeq, beq) which as before uses A and b for the $\leq$ constraints and a second pair Aeq, beq for the equality constraints. Use the following sequence to find a feasible solution.

```
% equality constraints
Aequals = [ 2 0 4 1 0 0; 1 1 1 0 1 0; 0 −2 1 0 0 1; 0 0 0 0 1 0 ]
equals = [ 4  3  −2  0 ]
% inequality constraints, all ≥ 0 except x(5)
A = eye(6); % eye returns a unit diagonal matrix, in this case 6 × 6
A = −A; A( 5, 5 ) = 0
b = zeros(6,1) % zeros returns a zero matrix, in this case 6 rows, 1 columns
% objective, minimise x(5) − x(6)
f = [  0  0  0  0  1 −1 ]
X = linprog(f, A, b, Aequals, equals)
```

7. Consider once again the bed-making factory of Exercise 4. Following market surveys the factory is considering switching production from its Ortho model to a new Rest-Rite model. Whereas the material figures for Ortho were 12, 7 and 15, the corresponding figures for the Rest-Rite are 12 (unchanged), 16 and 10, and the retail value is £550. Assuming that it is not feasible to make incomplete beds the decision whether or not to switch production can be modelled as follows:

   Let d be the decision variable defined by

$$\begin{array}{ll} d = 1: & \text{switch from Ortho to Rest-Rite} \\ d = 0: & \text{stay with Ortho.} \end{array}$$

Let x_1 denote the number of Sleep-eezy beds to be made, x_2 the number of Ortho beds, x_3 the number of Rest-Rite beds and let z denote total retail value. The linear programming problem becomes

$$\begin{aligned} \text{maximise} \quad & z = 200x_1 + 400x_2 + 450x_3 \\ \text{subject to} \quad & 5x_1 + 12x_2 + 12x_3 \le 120 \\ & 10x_1 + 7x_2 + 16x_3 \le 140 \\ & 12x_1 + 15x_2 + 10x_3 \le 180 \\ & x_2 \le 16(1-d) \\ & x_3 \le 16d \\ & d \le 1 \\ & x_1, x_2, x_3, d \ge 0, \quad x_1, x_2, x_3, d \quad \text{integer.} \end{aligned}$$

The number 16 is chosen as being larger than any possible value of the problem variables. Solve the problem using *bintprog* as explained in question 5 and illustrated below. Use a Matlab program of similar form, to that shown below. Noting that the decision variable d is already in single digit binary form, represent each of the problem variables, x_1, x_2 and x_3 as strings of four binary digits.

```
u = [ 1  2  4  8 ];
% Use the continuation mark ... for multi-line input
A = [ 5*u  12*u  12*u  0;  10*u  7*u  16*u  0;  12*u  15*u  10*u  0; ...
      0*u  1*u  0*u  16; 0*u  0*u  1*u  −16 ];
b = [  120;  140;  180;  16;  0 ];
f = [ −200*u  −400*u  −550*u  0 ];
x = bintprog( f, A, b )
```

Deduce that the factory should make the switch and that retail value may be increased by £600.

8. Use *bintprog* to solve the car hire company problem discussed in the problem of Sect. 7.7. In this example all the variables are binary and so there is no need for coversions to and from decimal. You may find the $d_1 = 1$, $d_2 = 1$, $d_3 = 0$, $d_4 = 0$, $d_5 = 1$, $d_6 = 0$, indicating that models 3, 4 and 6 are no longer required. In this example there are three other solutions, which could be found on the branch and bound tree.
9. Use *bintprog* to solve the travelling salesman problem of Sect. 7.7. The problem has 25 variables d_{ij} which may be represented by a single vector in which the variables are held in the order

$$d_{1,1}, d_{1,2}, \ldots, d_{1,5}, d_{2,1}, d_{2,2}, \ldots, d_{2,5}, d_{3,1}, \ldots$$

The problem as stated has no inequality constraints, the absence of which may be indicated by entries [ ] in the appropriate positions in the extended parameter list of *bintprog*. Typically a command would be of the form

```
[X  value] = bintprog( z, [ ], [ ], A, b );
```

which returns the solution variables in X and the value of the objective z at the solution in *value*. In this example the matrix of constraints has 10 rows and 25 columns. The task of entering all 250 entries in A may be simplified by noting that vector entries may be used to enter a particular sequence of values, but whichever way is chosen the task of entering the data correctly is not to be under estimated.

You may find that the program produces the solution

$$d_{12} = 1, \qquad d_{21} = 1, \qquad d_{35} = 1, \qquad d_{43} = 1, \qquad d_{54} = 1$$

which represents Manchester–Leeds–Manchester, and in a separate loop, Stoke–Preston–Liverpool–Stoke. Clearly this is not acceptable and so we re-solve the problem with the added (inequality) constraint

$$d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25} \geq 1$$

which has the effect of forcing at least one direct link between Manchester or Leeds and Stoke, Liverpool or Preston.

This added constraint produces a solution

$$d_{12} = 1, \qquad d_{25} = 1, \qquad d_{31} = 1, \qquad d_{43} = 1, \qquad d_{54} = 1$$

which either way round is an optimal route, namely Manchester–Leeds–Preston–Liverpool–Stoke–Manchester, a total distance of 250 miles.

Chapter 8
Optimisation

Aims In this chapter we look at methods for finding the maximum (or minimum) of a function of one or more variables, possibly subject to constraints. In so doing we generalise the linear problems of Chap. 7.

In particular we

- explain how we can regard the problem as one of minimisation.
- distinguish between local and global minima.
- describe the simple grid search and the golden section search methods for functions of a single variable.
- describe the method of steepest descent and a rank-one method for unconstrained problems involving functions of several variables.
- describe penalty function methods for constrained problems.

Overview Optimisation is concerned with finding a local maximum or minimum of a function of several variables $f(\mathbf{x}) \equiv f(x_1, x_2, \ldots, x_n)$. We examine methods for both **unconstrained** problems, in which the independent variables are free to take any values, and **constrained problems** in which the independent variables are constrained in some manner. We may have **simple bounds** such as $0 \le x_1 \le 1$ and $x_2 \ge 4$, linear constraints as in linear programming problems, or more complicated **nonlinear constraints** such as $x_1^2 + x_2 \le 1.5$, where x_1 and x_2 are independent variables. Unlike the linear problems of Chap. 7 it is not possible to give all embracing methods with guarantees to solve a wide class of problems. The general problem we investigate in this chapter is altogether more complex and for this reason we approach the topic by describing methods for

- unconstrained problems
- constrained problems.
- methods for functions of a single variable
- methods for functions of several variables.

Acquired Skills After reading this chapter and completing the exercises you will be able to

- formulate an optimisation problem as a minimisation problem.
- decide when to use and how to apply the grid search and golden section search methods.
- decide when to use and how to apply the method of steepest descent and a rank-one method, and have some idea as to how a rank-two method might be formulated.
- understand what is meant by a penalty function.
- find the Lagrangian of a constrained problem.
- use the multiplier penalty function method on a Lagrangian to find a solution to a constrained problem.

8.1 Introduction

Although we look at methods for finding optimal values we simplify the discussion by assuming that we are trying to find minimum, as opposed to maximum, values. This is the opposite situation to linear programming, where unless otherwise stated it is assumed that we are aiming to find a maximum. However, by the reasoning given in the overview of Chap. 7 neither case is restrictive since if we are required to find the maximum value of a function f, we can do so by finding the minimum value of $-f$ and vice versa.

Consideration of what we mean by a **local minimum** and a **global minimum** leads to the distinction between a local minimum as a minimum with respect to a local region, whereas a global minimum is the minimum of all local minima. The methods we describe will only provide a local minimum. To decide if such a minimum is a global minimum is a separate issue. In practice we look for clues in the general shape of the function. Figure 8.1 shows a function, $f(x)$ of a single variable

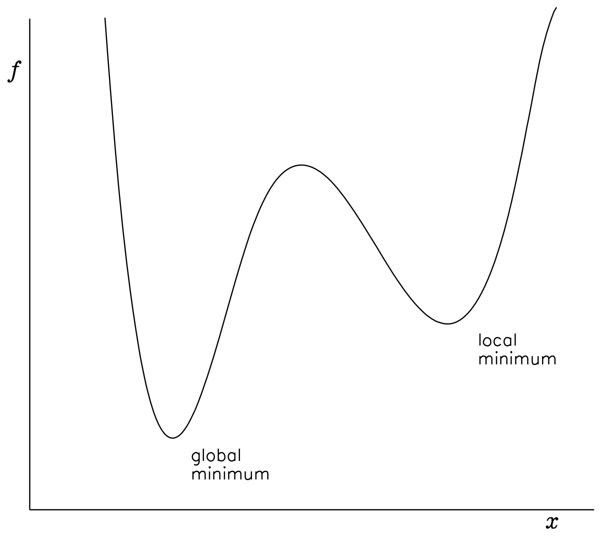


Fig. 8.1 Local and global minima

having a local minimum and what appears to be a global minimum, although more evidence from ranges of x not shown might be needed to decide if this is indeed the case.

8.2 Grid Searching Methods

We begin by considering methods for finding a local minimum of a function of a single variable based on searching a given interval. We assume that we are given an interval $[a, b]$ in which a local minimum exists. The methods we describe are based on dividing this interval into smaller sub-intervals. By comparing function values at the end-points of each sub-interval we decide which sub-interval contains a local minimum. This narrowing down process is repeated until f and/or x at the local minimum are found to the required accuracy.

8.2.1 Simple Grid Search

We choose to divide the current interval containing the minimum into a further four sub-intervals of equal length. There are other possibilities, but this particular choice is both easy to implement and reasonably efficient in terms of the number of function evaluations required. The method is illustrated in the following problem.

Problem

In the piston arrangement shown in Fig. 8.2 it can be shown that the rate of change of x with respect to θ is given by

$$\frac{dx}{d\theta} = r\sin\theta + \frac{r^2\sin 2\theta}{2\sqrt{l^2 - r^2\sin^2\theta}}.$$

Using a simple grid search find the maximum value of $\frac{dx}{d\theta}$ for θ in the interval $[0, \pi]$. Find a solution correct to 5 decimal places. Assume $r = 1$ and $l = 3$.

Solution

We convert the problem to one of minimisation. Introducing $f = -\frac{dx}{d\theta}$ we look for a minimum of $f = f(\theta)$ for θ in the interval $[0, \pi]$.

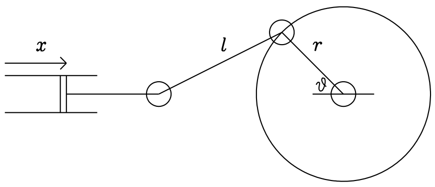


Fig. 8.2 Piston

Evaluating f at five equally-spaced points in the interval $[0, \pi]$ including the end points, we have

θ	0	0.78540	1.57080	2.35619	3.14159
$f(\theta)$	0	−0.87861	−1	−0.53561	0

The interval containing a minimum can be reduced to [0.78540, 2.35619]. Performing a further sub-division we have

θ	0.78540	1.17810	1.57080	1.96350	2.35619
$f(\theta)$	−0.87861	−1.04775	−1	−0.80001	−0.53561

Note that this second step just involves two new function evaluations, since we already have $f(\theta)$ at the two end points and at the middle point $\theta = 1.57080$. The interval can now be reduced to [0.78540, 1.57080], from which another set of evaluations produces

θ	0.78540	0.98175	1.17810	1.37445	1.57080
$f(\theta)$	−0.87861	−0.99173	−1.04775	−1.04827	−1

Proceeding in this way, after a further 6 iterations we obtain

θ	1.27014	1.27320	1.27627	1.27934	1.28241
$f(\theta)$	−1.05461	−1.05463	−1.05464	−1.05464	−1.05462

indicating that f has a local minimum of -1.05464 (to 5 decimal places) at some point in the interval [1.27320, 1.27934].

It so happens that in this example the values of the function are the first to converge to the required accuracy. To pinpoint the position of the minimum to a similar accuracy we continue to narrow the interval. Following a further 9 iterations we find

θ	1.27715	1.27715	1.27715	1.27715	1.27715
$f(\theta)$	−1.05464	−1.05464	−1.05464	−1.05464	−1.05464

indicating that the local minimum is at $\theta = 1.27715$ (to 5 decimal places). In conclusion, we have $\frac{dx}{d\theta}$ reaching a maximum value of 1.05464 for $\theta = 1.27715$, (approximately 73°).

Discussion

The details of the simple grid search are specified in Table 8.1. The method is not dissimilar to the bisection method of Chap. 3 in which we also successively halve the interval at each step. In the grid search the interval containing the local minimum is halved after 2 function evaluations. However we can do better than this. In real time applications in which the function is time consuming to evaluate such considerations can be important.

Table 8.1 Simple grid search

	Find a local minimum of $f(x)$ in the interval $[a, b]$ using a simple grid search
1	Locate an interval $[a, b]$ containing a minimum
2	Evaluate function values at the points a, $(a+b)/4$, $(a+b)/2$, $3(a+b)/4$, b two of which will be already known from the previous iteration
3	Find three consecutive points that form a sub-interval containing a minimum. Set a and b to the end points of the sub-interval
4	Test for convergence
5	Repeat from step 2

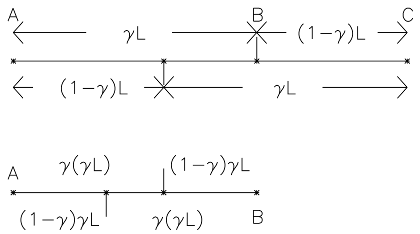


Fig. 8.3 Golden section search

8.2.2 Golden Section Search

In terms of reducing the number of function evaluations the **golden section search**, which divides the current interval in the ratio $1-\gamma : \gamma$, where γ satisfies $\gamma^2+\gamma=1$ ($\gamma \approx 0.618$) is a more efficient method. A typical sequence is shown in Fig. 8.3 which assumes that on consideration of the divided interval [A,C] the local minimum is found to lie in the sub-interval $[A, B]$. The ratio $1-\gamma : \gamma$, which is used intuitively by architects and artists, is considered to be aesthetically pleasing, hence the term *golden section*. The performance of the method is illustrated using the previous problem.

Problem

Use the golden section search to find a minimum value of $f = f(\theta)$ for θ in the interval $[0, \pi]$, where

$$f(\theta) = -\sin\theta - \frac{\sin 2\theta}{2\sqrt{9-\sin^2\theta}}.$$

Find a solution correct to 5 decimal places for both θ and $f(\theta)$.

Solution

Dividing the interval $[0, \pi]$ by golden section we have intermediate points $\pi \times (1-\gamma)$ and $\pi \times \gamma$, that is 1.19998 and 1.94161 respectively. Evaluating the function at

Table 8.2 Golden section search

	Find a local minimum of $f(x)$ in the interval $[a, b]$ using the golden section search
1	Locate an interval $[a, b]$ containing a minimum of f
2	Evaluate function values at the points $a, a + (1 - \gamma)(b - a), a + \gamma(b - a), b$ three of which will be already known from the previous iterations
3	Find three consecutive points that form a sub-interval containing a minimum. Set a and b to the end points of the sub-interval
4	Test for convergence
5	Repeat from step 2

these points and the end points, we have

θ	0	1.19998	1.94161	3.14159
$f(\theta)$	0	−1.05048	−0.81359	0

The interval containing a minimum can be reduced to [0, 1.94161]. Dividing this interval in golden section we have intermediate points 0.74163 and 1.19998. It is a feature of division by golden section, which may be verified analytically, that one point is retained in successive divisions. From just one further function evaluation we obtain

θ	0	0.74163	1.19998	1.94161
$f(\theta)$	0	−0.84589	−1.05048	−0.81359

The interval containing the minimum has now been reduced to [0.74163, 1.94161]. Repeating the process we obtain

θ	0.74163	1.19998	1.48326	1.94161
$f(\theta)$	−0.84589	−1.05048	−1.02695	−0.81359

and after a further 10 iterations

θ	1.27288	1.27661	1.27891	1.28264
$f(\theta)$	−1.05463	−1.05464	−1.05464	−1.05462

indicating that the minimum value of f is −1.05464 (to 5 decimal places). A further 13 iterations show that this minimum value is found at $\theta = 1.27715$ (to 5 decimal places).

Discussion

The details of the golden section search are specified in Table 8.2. At each step the interval containing the local minimum is reduced by a factor of 0.6 after one function evaluation. Two function evaluations give a reduction by a factor of 0.4 (i.e. 0.6×0.6), which is an improvement on the simple grid search as this gives a reduction of 0.5 after two function evaluations. Since function evaluations are likely to be the most time consuming part of either method the golden section search is to be recommended.

8.3 Unconstrained Optimisation

We now turn our attention to functions of more than one variable. Grid searching methods will not be considered as we have already discussed methods for a single variable. The extension from functions of a single variable to functions of more than one variable follows quite naturally, if a little cumbersome to implement. Instead we look at methods which take a more informed and therefore more direct route to a minimum. If for example, we are looking for a local minimum of a function of two variables the problem may be visualised as that of finding a low point of a mountain range. As with the one-dimensional problem considered in the previous section a local minimum is not necessarily a global minimum. Although problems involving functions of more than two variables are more difficult to visualise, the mountain range model does suggest a general strategy.

The methods we describe are based on the following strategy

1. Select a starting point.
2. Choose a direction in which to search.
3. Find a local minimum in this direction.
4. Move to this point.
5. If this seems to be the local minimum then stop, otherwise repeat from step 2.

The details of step 2 vary from method to method, but the other steps remain the same. We can think of the strategy in terms of a walker zig-zagging down a hillside in an attempt to reach the bottom of the valley. The situation is represented in Fig. 8.4. The curved lines represent contours, that is, lines of equal values of f.

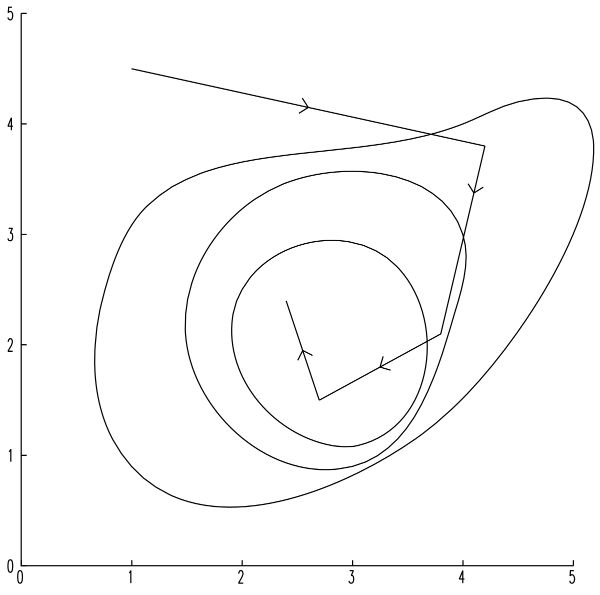


Fig. 8.4 Search for a local minimum

8.3.1 The Method of Steepest Descent

In the method of **steepest descent** we take the direction of search to be in the direction of the steepest *downhill* slope. In terms of the walker standing on the hillside and looking for the bottom of the valley this would seem to be a sensible line to take, at least in the short term. Admittedly the analogy breaks down if the walker is guided to walk off the edge of a cliff, but we assume that the function we are trying to minimise is reasonably smooth. Given $f = f(x, y)$ the slope of f has components $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$. It follows that the direction of steepest downhill slope at the point (a, b) is given by the vector in the opposite direction, namely $-\nabla f$, where $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ evaluated at the point (a, b). The method is illustrated by the following problem.

Problem

The perimeter P of the trapezium shown in Fig. 8.5 may be expressed in terms of the lengths of two sides x, y and the area A by the formula

$$P(x, y) = x + y + \sqrt{\left(\frac{4A}{x + y}\right)^2 + (y - x)^2}.$$

Use the method of steepest descent to find the values x and y which minimise P for $A = \frac{1}{4}$.

Solution

We proceed with the method of steepest descent using the general strategy outlined in Sect. 8.3.

1. Select a starting point.
   We choose $x = 0.25$ and $y = 1.0$, since $A = 0.25$ suggests that values for x and y of similar size are appropriate.

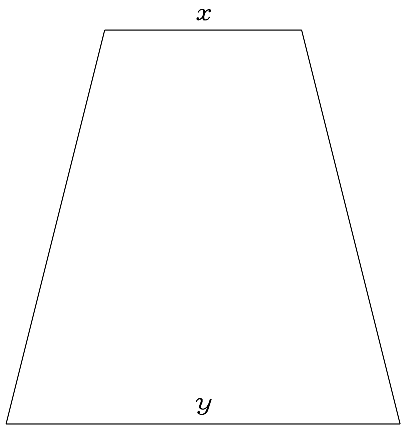


Fig. 8.5 Trapezium

Table 8.3 Steepest descent search for a minimum of $P(x, y)$

Step	x	y	$P(x,y)$	$\frac{\partial P}{\partial x}$	$\frac{\partial P}{\partial y}$
1	0.2500	1.0000	2.3466	−0.1508	1.2170
2	0.3037	0.5667	2.0490	−0.5097	−0.0634
3	0.4632	0.5866	2.0103	−0.0285	0.2285
4	0.4732	0.5062	2.0010	−0.0743	−0.0096
5	0.4959	0.5092	2.0001	−0.0031	0.0235
6	0.4971	0.5006	2.0000	−0.0082	−0.0012
7	0.4996	0.5010	2.0000	−0.0004	0.0024
8	0.4997	0.5002	2.0000	−0.0008	0.0002
9	0.5002	0.5001	2.0000	0.0006	0.0003
10	0.5001	0.5000	2.0000	0.0002	0.0000
11	0.5000	0.5000	2.0000	0.0001	−0.0000

2. Choose a direction in which to search.

   The partial derivatives of P are given by

$$\frac{\partial P}{\partial x} = 1 - \frac{\left(\frac{1}{x+y}\right)^3 + (y-x)}{\sqrt{\left(\frac{1}{x+y}\right)^2 + (y-x)^2}} \quad \text{and} \quad \frac{\partial P}{\partial y} = 1 - \frac{\left(\frac{1}{x+y}\right)^3 - (y-x)}{\sqrt{\left(\frac{1}{x+y}\right)^2 + (y-x)^2}}.$$

   At the point $x = 0.25$ and $y = 1$ these derivatives evaluate to -0.150845 and 1.21704 respectively. The direction of search is therefore $(0.150845, -1.21704)$.
3. Find a local minimum in this direction.

   A line through the point $x = 0.25$, $y = 1$ in the direction $(0.150845, -1.21704)$ may be expressed in the form

$$x = 0.25 + 0.150845t \tag{8.1}$$

$$y = 1 - 1.21704t. \tag{8.2}$$

   Using the golden section search we find a local minimum at $t = -0.3560$.
4. Move to this point

   We now have

$$x = 0.25 + 0.1508 \times 0.35560 = 0.3037$$

$$y = 1 - 1.2170 \times 0.35560 = 0.5667$$

   as new estimates for the local minimum of P.
5. If this seems to be the local minimum then stop, otherwise repeat from step 2.

   At a local minimum we expect the partial derivatives of P to be zero, or very near to zero. However at $x = 0.3037$ and $y = 0.5667$, $\frac{\partial P}{\partial x} = -0.5097$ and $\frac{\partial P}{\partial y} = -0.0634$ so it seems there is still some way to go. Accordingly we return to step 2 with these new values for x and y.

The sequence of results is summarised in Table 8.3, from which it can be seen that the iteration is terminated when the partial derivatives are less than 0.000005 in

Table 8.4 Method of steepest descent

	Find a local minimum of $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ using the method of steepest descent
1	Select a starting point **a**
2	Evaluate the direction of search, $-\nabla f_{\mathbf{a}} = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})_{\mathbf{x}=\mathbf{a}}$
3	Use the golden section search (or other method) to find a value of t to make $\mathbf{a} - t\nabla f_{\mathbf{a}}$ a local minimum
4	Use the value of t at the local minimum (**b**) in direction **s** from **a**, where $\mathbf{s} = -\nabla f_{\mathbf{a}}$
5	Test for convergence. Stop if the partial derivatives are sufficiently small and/or little or no difference in **a** and **b**
6	Replace **a** by **b** and repeat from step 2

magnitude, at which point $x = 0.5$, $y = 0.5$ and the value of the objective function P is 2. The trapezium becomes a rhombus.

Discussion

The method of steepest descent as described here may be generalised to deal with functions of more than two variables. The details are summarised in Table 8.4. Step 2 involves calculating derivatives. This may be achieved by supplying analytic derivatives as we did in the trapezium problem. However if such analytic expressions are not available, estimates may be provided using the techniques of Chap. 6.

Step 3 involves searching for the local minimum of a function of a single variable. In the problem regarding the trapezium we chose the interval $[-1, 1]$ and in general without prior investigation this should prove to be satisfactory. This interval could be widened in particular cases if no local minimum is forthcoming (see Exercise 1).

There are two tests for convergence shown in Table 8.4. If the sequence of estimates for the minimum shows little or no change, there is clearly no point in continuing. If at the same time the partial derivatives are all close to zero, we can be confident of having found a local minimum. If, however, not all derivatives are zero this might indicate that we are at some more exotic form of local minimum such as a saddle point, in hill climbing term we might be high on a ridge.

8.3.2 A Rank-One Method

The method of steepest descent is adequate but we can do better in terms of reducing the number of iterations. We achieve this by taking a broader view of the function. Whereas the method of steepest descent takes into account the first derivative of the function, a **rank-one method** takes into account first *and* second derivatives and in so doing attempts to follow the overall trend of the function rather than being preoccupied with the immediate local gradient.

We assume that we are at the point $x = a$, $y = b$ in our search for a local minimum of the function $f = f(x, y)$ and that in accordance with the strategy of Sect. 8.3 we search for a local minimum in the as yet unspecified direction (s_1, s_2). Using (8.1) and (8.2) as the model we write

$$x = a + ts_1$$
$$y = b + ts_2$$

in a Taylor series expansion of $f(x, y)$. For sufficiently small t we have

$$\begin{aligned} f(a + ts_1, b + ts_2) &= f(a, b) + t\left(s_1 \frac{\partial f}{\partial x} + s_2 \frac{\partial f}{\partial y}\right) \\ &\quad + \frac{t^2}{2}\left(s_1^2 \frac{\partial^2 f}{\partial x^2} + 2s_1 s_2 \frac{\partial^2 f}{\partial x \partial y} + s_2^2 \frac{\partial^2 f}{\partial y^2}\right) + \text{higher order terms} \end{aligned}$$

where all derivatives are evaluated at $x = a$, $y = b$. Our aim is to find values x, y such that $f(x, y)$ is less than $f(a, b)$. If we choose s_1 and s_2 so that

$$s_1 \frac{\partial^2 f}{\partial x^2} + s_2 \frac{\partial^2 f}{\partial x \partial y} = -\frac{\partial f}{\partial x} \tag{8.3}$$

$$s_1 \frac{\partial^2 f}{\partial x \partial y} + s_2 \frac{\partial^2 f}{\partial y^2} = -\frac{\partial f}{\partial y} \tag{8.4}$$

it follows that

$$\begin{aligned} f(x, y) &= f(a, b) \\ &\quad + \frac{1}{2}\left[(1 - t)^2 - 1\right]\left(s_1^2 \frac{\partial^2 f}{\partial x^2} + 2s_1 s_2 \frac{\partial^2 f}{\partial x \partial y} + s_2^2 \frac{\partial^2 f}{\partial y^2}\right). \end{aligned}$$

Therefore if $|t| < 1$ and $(s_1^2 \frac{\partial^2 f}{\partial x^2} + 2s_1 s_2 \frac{\partial^2 f}{\partial x \partial y} + s_2^2 \frac{\partial^2 f}{\partial y^2}) > 0$ then $f(x, y) < f(a, b)$. It is easy enough to ensure the first of these conditions by not wandering too far from the current estimate. The second condition is automatically satisfied at a local minimum, so there is reason to suppose that it will be satisfied if we are not too far away. Together the conditions suggest that $f(x, y) < f(a, b)$ for (x, y) local to (a, b) and so we have a local minimum.

Problem

Use a rank-one method to find values x and y which minimise $P = P(x, y)$, where

$$P = x + y + \sqrt{\left(\frac{1}{x + y}\right)^2 + (y - x)^2}.$$

Solution

This is the problem solved earlier using the method of steepest descent. As with steepest descent we follow the general strategy of Sect. 8.3.

Table 8.5 Estimation of second partial derivative of $P(x, y)$

n ($h = \frac{1}{2^n}$)	$\frac{\partial^2 P}{\partial x^2}$
1	0.788516
2	0.803356
$\vdots$	$\vdots$
9	0.824509
10	0.824605

1. Select a starting point.
   As before we begin at the point $\mathbf{x} = (0.25, 1.0)^T$.
2. Choose a direction in which to search.
   As we have already seen, at the point $x = 0.25$ and $y = 1$ we have $\frac{\partial P}{\partial x} = -0.150845$ and $\frac{\partial P}{\partial y} = 1.21704$. Equations (8.3) and (8.4) require second partial derivatives but rather than face the challenge of carrying out further analytic differentiation we use a numerical method from Chap. 6. For example, $\frac{\partial^2 P}{\partial x^2}$ can be found by evaluating

$$\frac{(\frac{\partial P}{\partial x})_{0.25+h,1.0} - (\frac{\partial P}{\partial x})_{0.25,1.0}}{h}$$

   and reducing h until a consistent result appears. We find components (see Table 8.5) at which point the difference between the latest estimates is less than 1.0×10^{-4}.
   By similar means we find estimates 0.43636 for $\frac{\partial^2 P}{\partial x \partial y}$ and 1.98948 for $\frac{\partial^2 P}{\partial y^2}$. We have

$$0.82461 s_1 + 0.43636 s_2 = 0.150845$$
$$0.43636 s_2 + 1.98948 s_2 = -1.21704$$

   from which we find $s_1 = 0.573166$, $s_2 = -0.737451$.
3. Find a local minimum in this direction.
   We find that P = P(t) = $(0.25 + 0.573166t, 1 - 0.737451t)$ has a local minimum at $t = 0.592808$.
4. Move to this point.
   We now have

$$x = 0.25 + 0.150845 \times 0.592808 = 0.589777$$
$$y = 1 - 1.21704 \times 0.592808 = 0.562833$$

   as new estimates for a local minimum of P.
5. If this seems to be a local minimum then stop, otherwise repeat from step 2.
   The first partial derivatives of P (0.278681 and 0.216599) are not zero and so we repeat from step 2.

Table 8.6 Rank-one search for a minimum of $P(x, y)$ using a linear equation solver

	x	y	$P(x, y)$	$\frac{\partial P}{\partial x}$	$\frac{\partial P}{\partial y}$
1	0.25000	1.00000	2.34659	−0.15084	1.21704
2	0.58978	0.56283	2.02062	0.27868	0.21660
3	0.50408	0.49539	2.00004	0.00765	−0.00972
4	0.50001	0.50001	2.00000	0.00004	0.00003
5	0.50000	0.50000	2.00000	0.00000	0.00000

The sequence of results is summarised in Table 8.6 from which it can be seen that the iteration is terminated when the partial derivatives are zero (to 5 decimal places), at which point $x = 0.5$, $y = 0.5$ and the value of the objective function P is 2. This compares with the result obtained by the method of steepest descent.

It is clear that the rank-one method has found the same local minimum of $P = P(x, y)$ in fewer iterations than the method of steepest descent. For most functions this is generally the case and so the extra computation involved in the rank-one method is deemed to be worthwhile.

8.3.3 Generalised Rank-One Method

Having considered the rank one method for finding a local minimum of a function of two variables we move on to the general case. We formulate the method and in so doing provide a more economical method. Extending (8.3) and (8.4) to deal with n functions $f_1, \ldots, f_n$ of n variables $x_1, \ldots, x_n$ we have

$$\begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2^2} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \cdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_3} & \frac{\partial^2 f}{\partial x_2 \partial x_3} & \frac{\partial^2 f}{\partial x_3^2} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \end{pmatrix} = \begin{pmatrix} -\frac{\partial f}{\partial x_1} \\ -\frac{\partial f}{\partial x_2} \\ -\frac{\partial f}{\partial x_3} \\ \vdots \end{pmatrix}.$$

Writing the above in matrix terms we have

$$\mathbf{Gs} = -\nabla f.$$

As an alternative to solving this linear system of equations in order to find the direction of search $\mathbf{s}$ we use the approximation

$$\mathbf{s} = -\mathbf{H}\nabla f \tag{8.5}$$

where $\mathbf{H}$ is as yet unspecified, but is chosen in such a way that it may be expected to converge towards the inverse of $\mathbf{G}$ as the search for the local minimum of f proceeds. This may be an unnecessary complication for a 2×2 system and similarly small systems, but if we are dealing with larger systems we replace finding a solution

Table 8.7 Rank-one method, without a linear equation solver

	Find a local minimum of $f(\mathbf{x})$, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ using a rank-one method
1	Select a starting point **a** and let **H** be the identity matrix
2	Evaluate $\nabla f_{\mathbf{a}} = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})_{\mathbf{a}}$
3	Test for convergence, depending on the problem. Stop if the partial derivatives are sufficiently small and/or little or no difference in **a** and **b**
4	Search for a local minimum, (**b**) in direction **s** from **a**, where $\mathbf{s} = -\mathbf{H}\nabla f_{\mathbf{a}}$
5	Replace **H** by $\mathbf{H} + \mathbf{E}$ where $E = \alpha\mathbf{u}\mathbf{u}^T$, $\mathbf{u} = (\mathbf{b} - \mathbf{a}) - \mathbf{H}(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}})$ and $\alpha = 1/(\mathbf{u}^T(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}}))$. Replace **a** by **b**
6	Repeat from step 2

to a system of tens or perhaps hundreds of linear equations by the simpler matrix–vector multiplication, (8.5).

There are many choices for **H**, but typically the rank-one method initially sets **H** to **I**, the identity matrix, and with each iteration adds an increment **E**, where **E** is a matrix of the form

$$\mathbf{E} = \alpha\mathbf{u}\mathbf{u}^T. \tag{8.6}$$

The matrix **E** is said to be of rank-one since it has one linearly independent column (every column is a multiple of any one of the others), hence the name rank-one method.

Now for the choice of how **H** is to be incremented. We recall from Chap. 6 that given a function $f = f(x)$ we may make the approximation $f''(x) = (f'(b) - f'(a))/(b - a)$ for x in the interval $[a, b]$. This result generalises to functions of several variables to give the approximation

$$\mathbf{G}(\mathbf{b} - \mathbf{a}) = \nabla_{\mathbf{b}} - \nabla_{\mathbf{a}}.$$

It follows that if we have an estimate **a** for the minimum and the search direction leads to an estimate **b** and in so doing **H** takes a new value $\mathbf{H} + \mathbf{E}$ we have

$$(\mathbf{H} + \mathbf{E})(\mathbf{b} - \mathbf{a}) = \nabla_{\mathbf{b}} - \nabla_{\mathbf{a}}.$$

Equating $\mathbf{H} + \mathbf{E}$ to $\mathbf{G}^{-1}$ we have

$$\begin{aligned}\mathbf{b} - \mathbf{a} &= (\mathbf{H} + \mathbf{E})(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}})\\ &= (\mathbf{H} + \alpha\mathbf{u}\mathbf{u}^T)(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}})\end{aligned}$$

which we can satisfy by choosing α and **u** so that

$$\mathbf{u} = (\mathbf{b} - \mathbf{a}) - \mathbf{H}(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}}) \tag{8.7}$$

$$\alpha\mathbf{u}^T(\nabla_{\mathbf{b}} - \nabla_{\mathbf{a}}) = 1. \tag{8.8}$$

We now have our algorithm for updating **H** at each step of the iteration. The rank-one method is summarised in Table 8.7.

In conclusion we note that although the method works well, particularly if the iteration is started with a good estimate of the solution, it cannot be guaranteed to converge. A method based on incrementing **H** by a rank two matrix, **E** constructed so as to ensure the search for a minimum proceeds in a downhill manner is more likely to converge.

Problem

Use the generalised rank-one method to find values x and y which minimise $P = P(x, y)$, where

$$P = x + y + \sqrt{\left(\frac{1}{x+y}\right)^2 + (y-x)^2}.$$

Solution

This is the problem solved earlier by the method of steepest descent and our first rank-one method. We use the generalised rank-one method to illustrate how the iteration proceeds without having to solve a set of linear equations at each step.

1. Select a starting point.

   As before we begin at the point $\mathbf{x} = (0.25, 1.0)^T$. We take **H** to be the matrix $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Since at this stage we have no idea of the value of **H** at the minimum point, the choice of the identity matrix would seem to be as good a starting point as any other.
2. Choose a direction in which to search.

   As we have already seen, at the point $x = 0.25$ and $y = 1$ we have $\frac{\partial P}{\partial x} = -0.150845$ and $\frac{\partial P}{\partial y} = 1.21704$, and so from (8.5) the search direction $\mathbf{s} = (s_1, s_2)^T$ is given by

$$\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = -\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} -0.150845 \\ 1.21704 \end{pmatrix}.$$

   The direction of search is therefore $(0.150845, -1.21704)^T$, which, given the particular starting choice for **H**, is the same as for steepest descent.
3. Find a local minimum in this direction.

   As before we find that $\tilde{P}(0.25 + 0.150845t, 1 - 1.21704t)$ has a local minimum at $t = 0.355955$.
4. Move to this point.

   We now have

$$x = 0.25 + 0.150845 \times 0.355955 = 0.303694$$
$$y = 1 - 1.21704 \times 0.355955 = 0.566789$$

   as new estimates for a local minimum of P.
5. If this seems to be a local minimum then stop, otherwise repeat from step 2.

   The partial derivatives of P (-0.50965 and -0.06317) are not zero and so we repeat from step 2 after completing the following step

Table 8.8 Rank-one search for a minimum of $P(x, y)$ without a linear equation solver

	x	y	$P(x, y)$	$\frac{\partial P}{\partial x}$	$\frac{\partial P}{\partial y}$
1	0.25000	1.00000	2.34659	−0.15084	1.21704
2	0.30369	0.56681	2.04901	−0.50961	−0.0608
3	0.49944	0.51194	2.00021	0.00982	0.03509
4	0.50176	0.49973	2.00000	0.00500	0.00095
5	0.49994	0.49998	2.00000	−0.00001	0.00005
6	0.50000	0.50000	2.00000	0.00000	0.00000

6. Update **H**

   Using (8.7) to find **u** and (8.8) to find α we have

$$\Delta_{\mathbf{x}} = \begin{pmatrix} 0.303694 - 0.25 \\ 0.566789 - 1.0 \end{pmatrix} = \begin{pmatrix} 0.053694 \\ -0.433211 \end{pmatrix}$$

   and since $\frac{\partial P}{\partial x}$ and $\frac{\partial P}{\partial y}$ at $x = 0.25,\ y = 1.0$ are respectively -0.150845 and 1.21704 we have

$$\Delta_{f'} = \begin{pmatrix} -0.509648 + 0.150845 \\ -0.063168 - 1.21704 \end{pmatrix} = \begin{pmatrix} -0.35880 \\ -1.28021 \end{pmatrix}.$$

   Therefore

$$\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = -\begin{pmatrix} 0.053694 \\ -0.433211 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -0.35880 \\ -1.28021 \end{pmatrix}.$$

   Thus $u_1 = 0.41250$, $u_2 = 0.84699$, and so by (8.8)

$$\alpha = \frac{1}{[(0.41250, 0.84699) * (-0.35880, -1.28021)^T]}$$
$$= -0.81147.$$

   Using these values we have a new **H** given by

$$\mathbf{H} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + 0.81147 \begin{pmatrix} 0.41250^2 & 0.41250 \times 0.84699 \\ 0.41250 \times 0.84699 & 0.84699^2 \end{pmatrix}$$

   and so we continue from step 2.

The sequence of results is summarised in Table 8.8 from which it can be seen that the iteration is terminated when the partial derivatives are zero (to 5 decimal places) at which point $x = 0.5$, $y = 0.5$ and the value of the objective function P is 2. This compares well with previous results. The economies achieved in not having to solve a system of linear equations at each step outweighs the cost of the extra iteration.

8.4 Constrained Optimisation

We now turn our attention to finding the local minimum of a function for which the variables are constrained. The problem is not unlike that of the linear programming of Chap. 7 although we now expect to deal with a nonlinear objective and allow for nonlinear constraints.

8.4.1 Minimisation by Use of a Simple Penalty Function

We consider a simplification in which the constraints on the objective are all equalities. We begin with a problem having just one equality constraint and introduce the **penalty function method**. It might be possible in such simple cases to eliminate one of the variables and solve the problem analytically. However, practical problems rarely present themselves in this manner and so we look for a more general method.

Problem

Find a local minimum of $f = f(x, y)$ where

$$f(x, y) = x^2 + xy - 1$$

subject to

$$x + y^2 - 2 = 0$$

using a simple penalty function method.

Solution

We solve the problem by finding a local minimum of a new function $\hat{f} = \hat{f}(x, y)$ where

$$\hat{f}(x, y) = x^2 + xy - 1 + \sigma\bigl(x + y^2 - 2\bigr)^2.$$

In this case $(x + y^2 - 2)^2$ is referred to as the **penalty function** since it is a measure of by how much the constraint on the problem is not satisfied. The hope is that in minimising $\hat{f}$ we will also be minimising the penalty function (to zero). Using the rank-one method of Sect. 8.3.2 with starting values $x = 1$, $y = 1$ and $\sigma = 1$ we find that $\hat{f}$ has a local minimum at $x = -1.03739$, $y = 1.78403$ and that the value of the penalty function is 0.02113.

Since the penalty function is not quite zero, we repeat but with a new $\hat{f}(x, y)$ defined by

$$\hat{f}(x, y) = x^2 + xy - 1 + \sigma\bigl(x + y^2 - 2\bigr)^2, \quad \sigma = 10,$$

in the hope that the value of the penalty function at the minimum will be smaller to minimise the effect of the factor 10. This time we find a local minimum at $x = -1.01645$, $y = 1.74099$ and that the penalty function has a value of order 10^{-4}. We are therefore getting close to a zero penalty.

Table 8.9 Simple penalty function method, results

Local minimum of $f(x, y) = x^2 + xy - 1 + \sigma(x + y^2 - 2)^2$ for various σ

σ	Values at the minimum for a given σ			
	x	y	$f(x, y)$	$(x + y^2 - 2)^2$
1	-1.0374	1.7840	-1.7534	0.2×10^{-1}
10	-1.0165	1.7410	-1.7343	0.2×10^{-3}
100	-1.0143	1.7366	-1.7324	0.2×10^{-5}
1000	-1.0141	1.7362	-1.7322	0.2×10^{-7}
10000	-1.0141	1.7361	-1.7322	0.2×10^{-9}

Table 8.10 Simple penalty function method

Find a local minimum of $f(x_1, x_2, \ldots, x_n)$ subject to the m constraints

$$c_1(x_1, x_2, \ldots, x_n) = 0$$
$$\ddots$$
$$c_m(x_1, x_2, \ldots, x_n) = 0$$

1	Set $\sigma = 1$
2	Find a local minimum of $f + \sigma \sum_{i=1}^{m} c_i^2$
3	Increase sigma and find another local minimum
4	Test for convergence
5	Repeat from step 3

We repeat the process, increasing σ until the value of the penalty function is sufficiently close to zero. We approach the solution in this tentative manner since we do not know in advance how large a coefficient will prove to be effective. The results are shown in Table 8.9 from which we conclude that subject to the constraint $x + y^2 - 2 = 0$, $f(x, y) = x^2 + xy - 1$ has a local minimum at $x = -1.01408$, $y = 1.73611$ and that the value of f at this point is -1.73220. Higher values of the penalty coefficient σ introduce numerical difficulties into the computation.

Discussion

The simple penalty function method may be extended to functions of more than two variables and to problems involving more than one equality constraint. Table 8.10 has the details. However the method is not guaranteed to succeed in all cases. Table 8.9 indicates that the theoretical values of σ necessary to achieve convergence may become large and cause problems for computer solutions, in practice some experimentation may be necessary. In this example we have managed to solve the problem but this will not always be the case. Nevertheless the principle is a useful one which we can apply effectively to a modified form of the function and its equality constraints, as shown in the next section.

Table 8.11 Search for minimum of $f(\mu)$, secant method first step

μ	Minimum $\mathcal{L}(x, y, \mu)$			$x + y^2 - 2$
	x	y	$\mathcal{L}$	
0.4	−0.5333	0.6666	−1.9067	−2.0889
0.3	−0.9000	1.5000	−1.7350	−0.6500

8.4.2 Minimisation Using the Lagrangian

We return to the previous problem of finding a local minimum of

$$f(x, y) = x^2 + xy - 1$$

subject to

$$x + y^2 - 2 = 0.$$

The **Lagrangian** of this problem is defined to be the function $\mathcal{L} = \mathcal{L}(x, y, \mu)$ where

$$\mathcal{L}(x, y, \mu) = x^2 + xy - 1 + \mu\big(x + y^2 - 2\big)$$

and where μ is known as a **Lagrange multiplier**. The Lagrangian bears a similarity to the penalty function of the previous section and has a very useful property which is shown in the following example.

Problem

Find a value μ for which a local minimum (x, y) of the Lagrangian

$$\mathcal{L}(x, y, \mu) = x^2 + xy - 1 + \mu\big(x + y^2 - 2\big)$$

is such that $x + y^2 - 2$ is zero.

Solution

We can regard this problem as finding a root of the function, $f = f(\mu)$ which for a given μ returns the value $x + y^2 - 2$ where x and y minimise the Lagrangian function quoted above.

We use the secant method of Chap. 3 to determine the sequence of μ values and a rank-one method (for example) to find a local minimum of $\mathcal{L}(x, y, \mu)$ for a given μ. The secant method requires two initial estimates which quite arbitrarily we chose to be 0.4 and 0.3 and obtain the results in Table 8.11. We take a secant step as shown in Table 3.6 namely,

$$\mu = 0.4 - \frac{(-2.0889)(0.3 - 0.4)}{0.65 - (-2.08889)} \tag{8.9}$$

$$= 0.2548 \tag{8.10}$$

Table 8.12 Search for minimum of $f(\mu)$, secant method continued

μ	Minimum $\mathcal{L}(x, y, \mu)$			$x + y^2 - 2$	$x^2 + xy - 1$
	x	y	$\mathcal{L}$		
0.2548	−6.7291	13.2033	−2.3670	165.5991	−44.5658
0.2998	−0.9021	1.5045	−1.7349	−0.6387	−1.5434
0.2997	−0.9042	1.5088	−1.7348	−0.6279	−1.5466
0.2897	−1.0578	1.8259	−1.7325	0.2762	−1.8125
0.2927	−1.0030	1.7133	−1.7322	−0.0676	−1.7124
0.2921	−1.0131	1.7342	−1.7322	−0.0058	−1.7305
0.2921	−1.0141	1.7362	−1.7322	0.0002	−1.7323
0.2921	−1.0141	1.7361	−1.7322	0.0000	−1.7322

to find a new value for μ in order to drive down the value of $x + y^2 - 2$ to zero.

Table 8.12 shows this latest estimate to be wide of the mark but it can be seen from the table of values of $x^2 + y^2 - 2$ that the secant method recovers. The results are interesting. We have found an x and y which as confirmed by earlier results is a local minimum of the function $f(x, y) = x^2 + xy - 1$ subject to the constraint $x + y^2 - 2 = 0$.

8.4.3 The Multiplier Function Method

The result from the example generalises. It can be confirmed analytically that $f(x, y)$ subject to a constraint has the same local minimum as its Lagrangian for a particular multiplier μ. In the next problem we exploit this property by finding a local constrained minimum of f by applying the simple penalty function method of Sect. 8.4.1 to the Lagrangian. The immediate problem is that of choosing an appropriate value of μ, but we are able to work towards it in an iterative manner. The whole process is known as the **multiplier penalty function** method and is illustrated in the following problem.

Problem

Find a local minimum of $f = f(x, y)$ where

$$f(x, y) = x^2 + xy - 1$$

subject to

$$x + y^2 - 2 = 0$$

using the multiplier penalty function method.

Solution

We look for local minima of

$$x^2 + xy - 1 + \mu\big(x + y^2 - 2\big) + \sigma\big(x + y^2 - 2\big)^2 \tag{8.11}$$

for increasing σ.

We begin by setting $\sigma = 1$ and $\mu = 0$. Using the rank-one method on (8.11) we obtain a local minimum at $x = -1.03573$ and $y = 1.78357$.

At a local minimum partial derivatives of the Lagrangian with respect to x and y should be zero and so if μ_{min} is the μ we are looking for we have

$$2x + y + \mu_{min} = 0$$
$$x + 2\mu_{min} y = 0.$$

But from (8.11) the partial derivatives with respect to x and y corresponding to the current μ, $\mu_{current}$, are given by

$$2x + y + \mu_{current} + 2\sigma\big(x + y^2 - 2\big) = 0$$
$$x + 2\mu_{current} y + 4\sigma y\big(x + y^2 - 2\big) = 0$$

and so we iterate towards μ_{min} using the rule

$$\mu_{next} = \mu_{current} + 2\sigma\big(x + y^2 - 2\big) \tag{8.12}$$

where $x + y^2 - 2$ (the constraint) is evaluated at the current minimum.

Using (8.12) the next value of μ is given by -0.28955. and applying the rank-one method to (8.11) with $\sigma = 1$ and $\mu = -0.28955$ we obtain a local minimum at $x = -1.03739$, $y = 1.78403$. From (8.12) the next value of μ is found to be 0.29074 and so the process continues for $\sigma = 1$ until we converge on the value of 0.29206 for μ_{min}.

The next step is to repeat the whole process with $\sigma = 10$. The results are shown in Table 8.13, from which it can be seen that no further increase in σ is necessary.

Table 8.13 Multiplier penalty function search for a minimum

Local minimum of $x^2 + xy - 1 + \mu(x + y^2 - 2) + \sigma(x + y^2 - 2)^2$

σ	μ	Values at the minimum for given σ and μ			
		x	y	$f(x, y)$	$(x + y^2 - 2)^2$
1	0	-1.03739	1.78403	-1.75342	2.1e-2
1	0.29074	-1.01419	1.73633	-1.73220	4.3e-7
1	0.29205	-1.01408	1.73611	-1.73220	9.5e-12
1	0.29206	-1.01408	1.73611	-1.73220	2.8e-16
10	0.29206	-1.01408	1.73611	-1.73220	1.5e-15
10	0.29206	-1.01408	1.73611	-1.73220	2.9e-19

Table 8.14 Multiplier penalty function method

	Find a local minimum of $f(x_1, x_2, \ldots, x_n)$ subject to the m constraints $c_1(x_1, x_2, \ldots, x_n) = 0$ $\ddots$ $c_m(x_1, x_2, \ldots, x_n) = 0$
1	Set $\sigma = 1$, $\mu_i = 0$ $(i = 1, \ldots, m)$
2	Find a local minimum of $f + \sum_{i=1}^{m} \mu_i c_i + \sigma \sum_{i=1}^{m} c_i^2$
3	Replace μ_i by $\mu_i + 2\sigma c_i$, where the c_i are evaluated at the minimum Find another local minimum
4	If the differences between successive minima are not sufficiently small then repeat from step 3
5	Increase σ and repeat from step 2
6	If the differences between successive minima are sufficiently small then stop
7	Repeat from step 3

Discussion

The multiplier penalty function method may be extended to functions of more than two variables and subject to more than one equality constraint. Table 8.14 has the details. The advantage of the method is that it generally avoids the problems of large σ values which are often found when using the simple penalty function method on the function to be minimised rather than its Lagrangian. A comparison of Tables 8.9 and 8.13 show how the multiplier penalty function method may avoid the problem of having to use very large σ values.

Summary In this chapter we have looked at ways of finding a local minimum of a function of one or more variables possibly subject to constraints. We began by considering what is meant by a local minimum and in particular established that finding a minimum is essentially the same problem as finding a maximum.

We first considered the simplest problem of all, namely that of finding a local minimum of a function of a single variable for which we used methods based on dividing a given interval. On grounds of efficiency and reliability the golden section search is the recommended method.

We then moved on to the problem of finding a local minimum of a function of several variables. We established a search strategy within which different choices for the direction of search give rise to different methods. We considered the method of steepest descent and a rank-one method, and whilst advocating the rank-one method it was realised that a rank-two method might be better. All the methods involve repeated searches in a given direction for which the golden section method is used.

Finally we considered the problem of finding a local minimum of a function of several variables subject to one or more constraints. We showed that it is possible to achieve this by means of a simple penalty function although using a penalty function applied to the Lagrangian, the so called multiplier penalty function method, is likely to be more reliable. Whatever the method the problem was solved by reducing the constrained problem to a series of unconstrained problems for which we had already established a rank-one solution method.

Exercises

1. Write a Matlab function with a heading such as

   $$\textit{function}[\textit{pos min}] = \textit{golden}(@f, a, b, \textit{eps})$$

   to find the minimum of a function f (specified in an M-file *f.m*) of a single variable in the range $[a, b]$ using the Golden Section Search. The search is to be continued until the absolute difference of successive approximations to the minimum position is less than *eps*. The outputs shown as *pos* and *min* are to contain the position of the minimum and its value. An optional third output parameter might be included to signal success or failure to find a minimum. Further options might include parameters to allow function values and relative values to control the iteration. Test the function using examples quoted in the chapter and on examples of your own choosing.

   As an added feature arrange for the *golden* function to extend the range $[a, b]$ by 50% at either the upper or lower end if the function values at a, b and the two intermediate points are in ascending or descending order. In such cases it would appear that the function does have a local minimum (or maximum) within the given range. This feature will be useful when the *golden* function is to be used as part of programs which implement optimisation routines involving line searches in which the search interval is not predetermined. Assuming an initial search interval of $[-1, 1]$, allowing for expansion is usually found to be successful.
2. Matlab provides a function *fminsearch* to find an unconstrained local minimum of a function, starting from a supplied point and continuing within prescribed convergence criteria until either a satisfactory solution is found or the search is abandoned. Taking default options the following command would find a local minimum of a function f defined in the M-file *f.m.* using the value stored in the variable *init* as the starting point.

   ```
   [pos min] = fminsearch(@fn, init)
   ```

   Default options may be overwritten using the *optimset* function as shown below.

   ```
   % Termination tolerances for x and f(x) values
   options = optimset('TolX',1e-5, 'TolF', 1e-6);
   [pos min] = fminsearch(@fn, init, options)
   ```

   Use *fminsearch* to verify results from the previous question and any of the examples quoted in the chapter.
3. Write a function *Pline* with parameter t to evaluate the function P given by

   $$P(x, y) = x + y + \sqrt{\left(\frac{1}{x+y}\right)^2 + (y-x)^2}$$

for (x, y) values given by $x = a + \lambda_1 t$, $y = b + \lambda_2$. Make a, b and the λ's available to *Pline* by separate global variables (see page 132 or the example below).

Use the function *golden* (from question 1) with *Pline* as a parameter to verify that a line through the point (0.25, 1) in the direction (0.15084, −1.21704) finds a local minimum P of 2.0490 for $t = -0.3560$ and so confirm the result quoted in Table 8.3.

4. Write a program to implement the method of Steepest Descent applied to the function, P specified in the previous question with a view to verifying results from the problem quoted in Table 8.3. Follow the scheme outlined in Table 8.4. The function P should be defined in a separate M-file to return the values of the function $P(x, y)$ and its partial derivatives $\frac{\partial P}{\partial x}$ and $\frac{\partial P}{\partial y}$ for any (x, y). Use the function *golden* with function *Pline* (from the previous) as a parameter to do the line searches for a minimum position.

```
% Global variables to be similarly defined in function Pline
  global est grad;
% initial estimate of minimum position
  est = [0.25 1];
% loop until the norm (size) of the increment
% to the current estimate is sufficiently small
  while norm(increment) > 0.1e-5
% obtain current function value and partial derivatives
  [f grad] = P(est);
% monitor progress
  s = sprintf('%8.4f', est, f, grad);
  disp(s)
% find the minimum along a line through a in direction -grad(a)
  grad = −grad
  [tvalue min] = golden (@Pline, −1, 1, 0.1e-7);
  increment = −tvalue∗grad;
  est = est + tvalue∗grad;
end
```

As an optional extra save successive estimates and so plot the progress of the search. Use the command *axis* ([0 1 0 1]) after the plot command to ensure equal scaling and observe that the minimum is approached in an orthogonal *staircase* manner, which is a feature of the method of steepest descent.

5. Modify the program from the previous question to implement the rank-one method, which does not require the solution of a set of linear equations. Use the method on the same function P(x,y). Follow the scheme outlined in Table 8.7 providing a replacement for function *Pline*, which has access to the matrix **H**. This will require an additional global variable to be declared in the new *Pline* function and in the main rank-one program **H** should be initialised to the 2 × 2 identity matrix. Verify the results shown in Table 8.8.

6. Use the program from question 5 to write the rank1 method as a general purpose function with the heading

$$function[position\ value] = rank1(fun, estimate)$$

where input parameter *fun* is the name of the function to be minimised and its partial derivatives. The function would be stored in a Matlab M-file with the name *fun.m* and supplied in the form @*name* when calling the function. The second parameter, *estimate* supplies an estimate of the minimum position. The actual position of the minimum and the value at the minimum are to be returned via output parameters *position* and *value*.

Test the program by using the function to verify the results shown in Sect. 8.4.1, which uses a simple penalty function to minimise a function subject to a constraint. Assuming the function $f(x, y) = x^2 + xy - 1 + \sigma(x + y^2 - 2)^2$ has been established as *fxy* in the Matlab-M file fxy.m with variable σ as global variable, the program might take the form

```
estimate = [1 1];
global sigma % sigma is also declared global in the M-file fxy.m
for i = 1:5;
  sigma = 10∧(i-1);
  [ minpoint minvalue] = rank1(@fxy, estimate);
  % monitor progress
  s = sprintf('%12d %10.5f %10.5f %10.5f', sigma, minpoint, minvalue);
  disp(s)
end
```

Ensure that the *rank*1 function may be written to deal with functions of any number n of variables using the heading quoted above. The value of n need not be passed as a parameter, it may be deduced internally from the size of the *estimate* vector using the command $[n\ m] = size(estimate)$; which returns the number of rows, n and the number of columns, m. However to allow for either row or column orderings the command $n = max(size(estimate))$; would be used to find n. A unit matrix **H** of order n may be established using the *eye* command, H = eye(n);

7. Consider the problem of finding a local minimum of the function

$$f(x_1, x_2, x_3) = x_1^4 + x_2 + x_3$$

subject to the constraints

$$\begin{aligned} x_1 + 3x_2^2 + 2x_3 - 1 &= 0 \\ 3x_1 \quad - x_2 + 4x_3 - 2 &= 0. \end{aligned}$$

Using the multiplier function method the function to be minimised is

$$\begin{gathered} x_1^4 + x_2 + x_3 \\ +\mu_1\big(x_1 + 3x_2^2 + 2x_3 - 1\big) + \mu_2(3x_1 - x_2 + 4x_3 - 2) \\ +\sigma\big((x_1 + x_2 + 2x_3 - 4)^2 + (x_1 + x_2^2 + x_3 - 3)^2\big). \end{gathered}$$

Write a program using either the *rank*1 function or the Matlab supplied function *fminsearch* to implement the multiplier function method as shown in Table 8.14. Table 8.15 indicates the results to be expected by setting the initial estimate of the minimum point as $= (1, 1, 1)$ and terminating the searches (stage 4) of the

Table 8.15 Exercise 7, results

Estimate of minimum position and value for various σ						
σ	μ_1	μ_2	x_1	x_2	x_3	$f(x_1, x_2, x_3)$
2	0.3530	−0.4264	0.6838	−0.6740	−0.1814	−0.6367
4	0.3530	−0.4262	0.6838	−0.6740	−0.1814	−0.6367
8	0.3526	−0.4265	0.6838	−0.6740	−0.1813	−0.6367
16	0.3526	−0.4253	0.6838	−0.6740	−0.1814	−0.6367
32	0.3526	−0.4270	0.6838	−0.6740	−0.1814	−0.6367
64	0.3531	−0.4274	0.6839	−0.6740	−0.1814	−0.6367

Table 8.16 Exercise 8, data

x	0	1	2	3	4	5	6	7
$f(x)$	2.5	1.7	1.3	1	0.8	0.6	0.4	0.2

Table 8.17 Exercise 8, results

α	β	$f(\alpha, \beta)$	$\frac{\partial f}{\partial \alpha}$	$\frac{\partial f}{\partial \beta}$
2.5	1	3.40330	−0.95939	3.80071
2.66753	0.33632	0.10160	0.57649	0.14353
2.60275	0.30911	0.08016	0.62894	−1.49994
2.44234	0.30395	0.03339	−0.00097	0.03059
2.44110	0.30344	0.03338	−0.00022	0.00054
2.44115	0.30344	0.03338	0	0

method as shown in Table 8.14 when the norm of the vector of difference of two consecutive estimates is less than 1.0e-7. It was found that more stringent tests would not produce convergence. It can be seen that we can be fairly confident of the results to three (and very nearly four decimal places). Further progress might be possible with some fine tuning of the program.

8. Find α and β so that the function $f(x) = \alpha e^{-\beta x}$ fits (in the least squares sense of Chap. 4) the data (Table 8.16) using the *rank*1 function. The problem is one of finding α and β which minimise the function

$$f(\alpha, \beta) = \sum_{i=0}^{7} \left[\alpha e^{-\beta x_i} - f(x_i)\right]^2$$

where the x_i and $f(x_i)$ are the data. Show that by taking 2.5 for α and 1 for β as estimates, the function *rank*1 takes the route (Table 8.17) (or similar) to a solution. Plot the data points and on the same graph as $\alpha e^{-\beta x}$ (with α and β as found by *rank*1) to give an indication of the accuracy of this approximating function.

9. Find a solution to the following system of nonlinear equations using an optimisation technique. The same problem was solved in Chap. 3 using Newton's method.

$$x\cos y + y\cos x = 0.9$$
$$x\sin y + y\sin x = 0.1.$$

Chapter 9
Ordinary Differential Equations

Aims In this chapter we look at a variety of numerical methods for solving ordinary differential equations. The aim is to produce solutions in the form of a table showing the values taken by the dependent variable for values of the one or more independent variables over given intervals.

Overview We distinguish two types of problem, namely

- the *initial-value problem*, in which the solution is required to meet prescribed conditions at one end of the interval.
- the *boundary-value problem*, in which the solution is required to meet conditions at both ends.

We start with the initial-value problem in the context of an equation of a very simple form known as a *first-order* equation and consider Euler's method as an introduction to the more powerful Runge–Kutta[1] methods. Both Euler and Runge–Kutta methods adopt a step-by-step approach in that the solution for the current value of the independent variable is advanced by an incremented value of the dependent variable. Typically we progress from a solution at x to a solution at $x + h$ until the whole range is covered. We show how more complicated equations may be reduced to a system of first-order equations to be solved by the same methods and so be in a position to solve the initial-value problem for any ordinary differential equation. We move on to consider the boundary-value problem for which we use either Runge–Kutta, varying the initial conditions until a solution which meets boundary conditions is found, or a method based on approximating the differential equation across the whole interval using approximations to the derivatives of the type discussed in Chap. 6. Throughout the chapter we compare the relative merits of the methods and conclude with a discussion of factors influencing the accuracy of the whole process of finding a numerical solutions to ordinary differential equations.

[1]Martin Kutta, engineer and mathematician, 1867–1944 introduced the method in his PhD thesis (1900), which was developed further by Carl Runge, physicist and mathematician, 1856–1927.

Acquired Skills After reading this chapter you will be able to decide if a given ordinary differential equation is an initial-value problem or a boundary-value problem. For the initial-value problem you will be able to use a Runge–Kutta method, if necessary reducing the original equation to a series of first-order equations. For the boundary-value problem you will be able to use what are known as *shooting* or *garden hose* methods to raise or lower initial estimates in order to find a solution, or use a method based on covering the area of interest with a grid of points.

9.1 Introduction

An ordinary differential equation is an equation involving variables and the *ordinary* (as opposed to partial) derivatives of some or all of those variables. Derivatives represent rates of changes and so differential equations often occur in the modelling of physical processes.

As was the case with integrals (Chap. 5) many of the differential equations which arise in practice do not have an analytic solution. Given an equation such as

$$\frac{dy}{dx} = f(x, y) \tag{9.1}$$

for a particular $f(x, y)$ we may not be able to write the relationship between y and x in an aesthetically pleasing and convenient form such as $y = x^2$ or $y = \sin x + \cos x$.

The equation

$$v\frac{dv}{dx} = -g \tag{9.2}$$

which models the downward velocity v of an object falling under gravity has the analytic solution

$$v^2 = C - 2gx \tag{9.3}$$

where x is the height above ground, C is a constant to be determined and g is the gravitational constant. If we are modelling free-fall from a height H, then $v(H) = 0$ and so $C = 2gH$. On the other hand the equation

$$\frac{dy}{dx} + \sin y = 1 + \sin x \tag{9.4}$$

which arises in the modelling of the fast-switching Josephson junctions of semiconductors does not have an analytic solution. If there is no analytic solution to the differential equation, or if the analytic solution does exist but is not apparent, we use a numerical method to provide a solution. In the case of an equation such as (9.4) we look to produce a table of y values corresponding to x values belonging to a given range.

The equations which we have quoted are all examples of *first-order* equations since only a first derivative is involved. The equation

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0 \tag{9.5}$$

which models the motion of a pendulum, is an example of a *second-order* equation. The order of a differential equation is defined by the order of the highest derivative.

In general, the solution to an nth-order differential equation contains n arbitrary constants which may be determined by requiring the solution to satisfy n independent conditions particular to the application. We have already seen an example of this in the first-order equation (9.2) and its solution (9.3). If the solution is analytic the arbitrary constants will be apparent, but even if an analytic solution does not exist, we are still required to provide sufficient independent conditions to determine the solution uniquely.

As a further example consider the second-order equation (9.5) which has the analytic solution

$$\theta = A\cos\omega t + B\sin\omega t$$

where $\omega = \sqrt{\frac{g}{l}}$ and A and B are constants to be determined. θ represents the angle through which the pendulum swings, l is the length of the pendulum and t is the measure of time. A and B may be uniquely determined by requiring the solution to satisfy the two conditions $\theta(0) = \theta_0$ and $\frac{d\theta}{dt}(0) = \theta_1$, where θ_0 and θ_1 are known values. These conditions model the release of the pendulum from angle θ_0 with angular velocity θ_1.

The problem of the falling body and the pendulum which we have just described are examples of an *initial-value problem*. An initial-value problem is a differential equation for which conditions relating to initial values of the independent variable are attached. As an initial-value problem (9.5) might be stated as

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0, \qquad \theta(0) = \theta_0, \qquad \frac{d\theta}{dt}(0) = \theta_1.$$

The same equation (9.5) may be used to model the buckling of a rod. Using a different notation we have

$$\frac{d^2y}{dx^2} + Ky = 0 \tag{9.6}$$

where x is measured along the rod, y is the lateral displacement and K is a constant related to the material properties of the rod. In this case the solution might be uniquely determined by requiring $y(0) = 0$ and $y(L) = 0$. These conditions are appropriate to a model of a beam of length L which is clamped at both ends. We now have what is known as a *boundary-value problem*. A boundary-value problem is a differential equation for which conditions relating to values of the independent variable at more than one point are attached. As a boundary-value problem (9.6) might be stated as

$$\frac{d^2y}{dx^2} + Ky = 0, \qquad y(0) = 0, \qquad y(L) = 0. \tag{9.7}$$

9.2 First-Order Equations

We begin by studying methods for finding solutions of first-order equations. Such equations are necessarily initial-value problems since only one condition, usually referred to as an *initial condition*, is required for a unique solution. Our problem in general terms may be stated as the initial-value problem

$$\frac{dy}{dx} = f(x, y), \qquad y(x_0) = y_0. \tag{9.8}$$

9.2.1 Euler's Method

Euler's method is based on the approximation of dy/dx at x by $(f(x+h) - f(x))/h$, where h (the *step-length*) is some suitably small value. This formula, which we employed in Sect. 6.2, is used in (9.8) to compute y at $x+h$ given y at x and so from an initial starting point the solution may be advanced through a given interval. The method is illustrated in the following problem.

Problem

Solve the following differential equation using Euler's method.

$$\frac{dy}{dx} = 3x - y + 8, \qquad y(0) = 3.$$

Compute the solution from 0 to 0.5 using a step-length of 0.1. Compare this solution with the analytic solution, $y = 3x - 2e^{-x} + 5$.

This initial-value problem is an example of a type which can occur in the modelling of changes in the population, y of a group of people, atomic particles, bacteria or whatever over time, x. In this example we assume that the population increases with respect to time but that this increase is countered with a decrease proportional to the size of the population. The rates of increase and decrease are $3x + 8$ and $-y$ respectively.

We assume that the model has been formed so that the solution measures the population in appropriate multiples, and that overall the population is sufficiently large to allow the use of real numbers with a fractional part, rather than whole integer numbers.

Solution

Euler's method advances the solution from x to $x+h$ using the approximation

$$y(x+h) \approx y(x) + h \left.\frac{dy}{dx}\right|_x \tag{9.9}$$

Table 9.1 Euler's method, step-length 0.1

x	y value by Euler's method	Correct y value	Error (absolute value)
0	3	3	0
0.1	3.5	3.49033	0.00967
0.2	3.98	3.96254	0.01746
0.3	4.442	4.41836	0.02364
0.4	4.8878	4.85936	0.02844
0.5	5.31902	5.28694	0.03208

Table 9.2 Euler's method, variable step-lengths

n $(h = 0.1/2^n)$	$y(0.5)$ Euler's method	Error (absolute value)
0	5.31902	0.03208
1	5.30253	0.01559
2	5.29462	0.00769
3	5.29076	0.00382
$\vdots$	$\vdots$	$\vdots$
11	5.28700	0.00006
12	5.28697	0.00003
13	5.28695	0.00001
14	5.28695	0.00001

and so for the current problem

$$y(x+h) \approx y(x) + h\big(3x - y(x) + 8\big). \tag{9.10}$$

We use the notation x_i, $i = 0, 1, \ldots, 5$ to denote the x values $0, 0.1, \ldots, 0.5$ and let y_i denote the approximation to $y(x_i)$. In this notation (9.10) becomes

$$y_{i+1} = y_i + h(3x_i - y_i + 8). \tag{9.11}$$

Using the initial conditions $x_0 = 0$, $y_0 = 3$ with $h = 0.1$ in (9.11) we have $y_1 = 3.5$. Using (9.11) again to calculate y_2 from y_1, we have $y_2 = 3.98$. Continuing in this way we eventually reach y_5. The results are shown in Table 9.1 where the error term is defined to be the difference between the value predicted by Euler's method and the correct value.

Discussion

It is evident from Table 9.1 that the error in the values predicted by Euler's method increase as the value of x increases. Although it may be proved analytically that errors do not get wildly out of hand, even so a step-length of 0.1 clearly is not giving very good accuracy. We can successively decrease the value of the step-length h until consistent values are obtained. Table 9.2 shows the results obtained for y at

$x = 0.5$ by successively halving h. Having obtained agreement for two successive values of h we would normally stop the calculation at this point and assume we have a valid result. It can be seen from the table that as h decreases linearly so does the error. This may be proved by analysis and so we may expect to obtain more and more accuracy using a smaller and smaller step-length. However, decreasing the step-length increases the number of arithmetic calculations and may mean that we are dealing with very small numbers, all of which is liable to increase the error in the computation. Indeed our calculations, shown in the table above, could lead us to accept a value of 5.28695 when the true value is 5.28694. In this case the difference may not matter very much, but in practice could be significant.

Although Euler's method may not be sufficiently accurate for some problems, the underlying idea is a good one and can be used to good effect in forming the more powerful Runge–Kutta methods.

9.2.2 Runge–Kutta Methods

Euler's method for solving the initial-value problem (9.8) advances the solution from x to $x+h$ using the approximating formula (9.9). As an alternative we use the same formula but with the derivative y' evaluated at the mid-point of x and $x+h$. It is envisaged that the derivative at the mid-point is more representative of the varying derivative across the range. The approximating formula becomes

$$y(x+h) \approx y(x) + hy'\left(x + \frac{h}{2}\right).$$

This approximation is the basis of the Runge–Kutta method, which is illustrated in the following problem.

Problem

Solve the initial-value problem

$$\frac{dy}{dx} = 3x - y + 8, \qquad y(0) = 3$$

using the simple Runge–Kutta method. Compute the solution from 0 to 0.5 using a step-length of 0.1. This is the same problem we used to illustrate Euler's method and is one to which we will return.

Solution

The method advances the solution from x to $x+h$ using an increment given by

$$hy\prime\left(x + \frac{h}{2}\right) = hf\left(x + \frac{h}{2}, y\left(x + \frac{h}{2}\right)\right) \quad \text{where } f(x, y) = 3x - y + 8.$$

Table 9.3 Simple Runge–Kutta, step-length 0.1

x	y value by simple Runge–Kutta	Correct y value	Error (absolute value)
0	3.00000	3.00000	0
0.1	3.49000	3.49033	0.00033
0.2	3.96195	3.96254	0.00059
0.3	4.41756	4.41836	0.00080
0.4	4.85840	4.85936	0.00096
0.5	5.28585	5.28694	0.00109

Table 9.4 Simpe Runge–Kutta, variable step-lengths

n ($h = \frac{0.1}{2^n}$)	$y(0.5)$ by simple Runge–Kutta	Error (absolute value)	Error $/h^2$
0	5.28585	0.00109	0.10902
1	5.28668	0.00026	0.10497
2	5.28687	0.00006	0.10301
3	5.28692	0.00002	0.10204
4	5.28693	0.00001	0.10156
5	5.28694	0.00000	0.10133
6	5.28694	0.00000	0.10121

Using

$$y\left(x+\frac{h}{2}\right)=y(x)+\frac{h}{2}y'(x)$$

the increment becomes

$$h\left(3\left(x+\frac{h}{2}\right)-\left(y+\frac{h}{2}(3x-y+8)\right)+8\right).$$

Starting with $x=0$, $y=3$ we obtain the results shown in Table 9.3.

Discussion

Clearly even at this early stage the simple Runge–Kutta method is more accurate than Euler's method. Repeating the calculations with successively decreasing values of the step-length h we have the results in Table 9.4. Whereas the results using Euler's method show a linearly decreasing error with linearly decreasing step-length h, in this case the errors decrease more rapidly. The final column indicates that the ratio of the error to h^2 is approximately constant. This confirms the theoretical result obtained by expanding the simple Runge–Kutta formula as a Taylor series that errors are likely to decrease in proportion to h^2. This is *quadratic* or *second-order* convergence. For this reason the simple Runge–Kutta method we have described is known as the second-order Runge–Kutta method and is the name that we use from now on. Euler's method has only *linear* or *first-order* convergence. Although for a given step-length the second-order Runge–Kutta method involves more calculation than Euler's method it is generally preferred because it has second-order, as opposed to linear, convergence.

9.2.3 Fourth-Order Runge–Kutta

With a little analysis it can be shown that the approximation

$$y(x+h) \approx y(x) + hy'\left(x + \frac{h}{2}\right)$$

which forms the basis of the second-order Runge–Kutta method may be re-stated with similar accuracy as

$$y(x+h) \approx y(x) + h\frac{dy}{dx} + \frac{h^2}{2}\frac{d^2y}{dx^2}$$

which will be recognised from the Taylor series expansion of $y(x+h)$. It is this level of agreement which establishes the theoretical validity of the second-order Runge–Kutta method.

Stated formally the second-order Runge–Kutta method advances the solution to the first-order differential equation $y'(x) = f(x, y)$ from $y(x)$ to $y(x+h)$ using

$$y(x+h) = y(x) + k_2$$

where

$$\begin{aligned} k_1 &= hf(x, y) \\ k_2 &= hf\left(x + \frac{1}{2}h, y + \frac{1}{2}k_1\right). \end{aligned}$$

However, with a little more effort we can do better by producing approximations that correspond as closely as possible to the Taylor series. We include derivatives evaluated at more points in the interval, $[x, x+h]$ and so produce more and more accurate Runge–Kutta methods.

The fourth-order Runge–Kutta method is probably the most widely used. The formula given below agrees with the Taylor series up to and including terms of order h^4 and may be used for advancing the solution from $y(x)$ to $y(x+h)$ using

$$y(x+h) = y(x) + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9.12}$$

where

$$\begin{aligned} k_1 &= hf(x, y) \\ k_2 &= hf\left(x + \frac{1}{2}h, y + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(x + \frac{1}{2}h, y + \frac{1}{2}k_2\right) \\ k_4 &= hf(x + h, y + k_3). \end{aligned}$$

Problem

Apply the fourth-order Runge–Kutta method to the initial-value problem

$$\frac{dy}{dx} = 3x - y + 8, \qquad y(0) = 3$$

using a step-length of 0.1.

Solution

Using the scheme shown above, we obtain the results in Table 9.5.

Discussion

The results are clearly more accurate than anything achieved previously. Repeating the calculations with successively decreasing values of the step-length h in a search for greater accuracy we obtain the results in Table 9.6. The almost constant nature of the ratio shown in the final column confirms that for a given step-length h the error in each estimated value is of order h^4. We must be aware that in applying any of the methods described in this chapter there is an accumulation of error during any computation. This error will result from *truncation error* and from *rounding error*. Truncation error is the error resulting from making approximations, for example by truncating the Taylor series expansion. Rounding error is the error incurred in the course of the, often numerous, arithmetic operations involved in applying any numerical method.

Table 9.5 Fourth-order Runge–Kutta, step-length 0.1

x	y by Runge–Kutta	Correct value y	Error (absolute value)
0	3	3	0
0.1	3.49033	3.49033	0.16×10^{-6}
0.2	3.96254	3.96254	0.30×10^{-6}
0.3	4.41836	4.41836	0.40×10^{-6}
0.4	4.85936	4.85936	0.49×10^{-6}
0.5	5.28694	5.28694	

Table 9.6 Fourth-order Runge–Kutta, variable step-lengths

n $(h = \frac{0.1}{2^n})$	Error at $x = 0.5$ (absolute value)	Error $/h^4$
0	0.55×10^{-6}	0.55×10^{-2}
1	0.33×10^{-7}	0.53×10^{-2}
2	0.20×10^{-8}	0.52×10^{-2}
3	0.12×10^{-9}	0.51×10^{-2}

All the methods we describe involve choosing an appropriate step-length, h. If we were progressively to decrease h, possibly by a factor of 0.5, and reach a situation at which solutions for successive values of h are in agreement to a certain accuracy, we would argue that accumulated error is not making a significant contribution and that we have a solution to the accuracy implied.

9.2.4 Systems of First-Order Equations

The methods for solving the initial-value problem for a single first-order equation are applicable to systems of first-order equations. Assuming we have values for all the dependent variables at the same initial value of the independent variable, we advance the solution for all the dependent variables at each step of the independent variable. We illustrate the process using the Runge–Kutta method.

Problem

Solve the equations

$$\frac{dw}{dx} = \sin x + y$$
$$\frac{dy}{dx} = -w + \cos x$$

for values of x in the interval $[0, 0.5]$. The initial conditions are $w(0) = 0$ and $y(0) = 0$. Use the fourth-order Runge–Kutta method with a step-length of 0.1. Compare the computed solution with the analytic solution, which is $w = x \sin x$, $y = x \cos x$.

Solution

The solutions for each equation are simultaneously advanced from the initial conditions. Using the fourth-order Runge–Kutta method we have

$$w_{n+1} = w_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9.13}$$

$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{9.14}$$

where in the case of (9.13)

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf\left(x_n + \frac{1}{2}h,\, y_n + \frac{1}{2}k_1\right) \\ k_3 &= hf\left(x_n + \frac{1}{2}h,\, y_n + \frac{1}{2}k_2\right) \\ k_4 &= hf(x_n + h,\, y_n + k_3) \end{aligned}$$

Table 9.7 Fourth-order Runge–Kutta, system of equations

x	w	y
0	0	0
0.1	0.00998	0.09950
0.2	0.03973	0.19601
0.3	0.08866	0.28660
0.4	0.15577	0.36842
0.5	0.23971	0.43879

with $f(x, y) = \sin x + y$. In the case of (9.14) we have

$$\begin{aligned}
k_1 &= hf(x_n, w_n)\\
k_2 &= hf\left(x_n + \frac{1}{2}h, w_n + \frac{1}{2}k_1\right)\\
k_3 &= hf\left(x_n + \frac{1}{2}h, w_n + \frac{1}{2}k_2\right)\\
k_4 &= hf(x_n + h, w_n + k_3)
\end{aligned}$$

with $f(x, w) = -w + \cos x$.

Using this system we eventually obtain the results in Table 9.7 and these values are correct to the number of decimal places shown.

9.2.5 Higher Order Equations

So far we have only considered first-order equations of the form $\frac{dy}{dx} = f(x, y)$. Not all the equations we are likely to meet in practice will be of this form, but we can reduce most higher order equations to a system of first-order equations.

For example the second-order equation

$$r\frac{d^2t}{dr^2} + \frac{dt}{dr} = 0$$

may be reduced to a system of first-order equations by introducing a new variable $z = z(r)$, where $z = \frac{dt}{dr}$.

We have

$$\begin{aligned}
r\frac{dz}{dr} + z &= 0\\
\frac{dt}{dr} &= z.
\end{aligned} \tag{9.15}$$

The method extends to higher orders. For example an equation involving $\frac{d^3y}{dx^3}$ could be reduced by introducing two new variables z and w, with $z = \frac{dy}{dx}$ and $w = \frac{dz}{dx}$. We are now in a position to attempt the solution of solving the initial-value problem for equations of any order.

9.3 Boundary Value Problems

We consider problems for which the solution is required to meet specified conditions at both ends of a range. Two methods are presented, a method that for obvious reasons is known as the *shooting* or *garden-hose* method and a method which approximates derivatives over the range using *difference equations*.

9.3.1 Shooting Method

Anyone who has used a garden hose, or has tried to hit any kind of target will know that there is a trial and error element involved. First attempts will either overshoot the target or fall short. However, this effort is not wasted since it usually gives an indication as to whether to raise or lower the aim. A similar scheme may be applied to boundary-value problems. We form an initial-value problem, which we solve by one of the methods described earlier, by assuming a set of initial conditions and see where the solution leads. We then vary the initial conditions until we hit the target at the other end. The following problem illustrates the method.

Problem

The following equation is an example of a type which arises in the modelling of the temperature, y of a cooling fin at distance x from a base position.

$$\frac{d^2y}{dx^2} = 2y.$$

Find a solution using the shooting method which satisfies $y = 1$ at $x = 0$ and $dy/dx = 0$ at $x = 1$. Tabulate the solution at the points $0.1, 0.2, \ldots, 1$. Compare the result with analytic solution namely,

$$c_1 e^{\sqrt{2}x} + c_2 e^{-\sqrt{2}x}$$

where for this example, $c_1 = 0.05581$ and $c_2 = 0.94419$.

Solution

Writing $z = dy/dx$ this second-order equation may be written as a system of first-order equations, namely

$$\frac{dz}{dx} = 2y$$
$$\frac{dy}{dx} = z$$

where $y = y(x)$ and $z = z(x)$ are such that $y(0) = 1$ and $z(1) = 0$.

Table 9.8 Boundary value problem, solution using shooting method

x	Predicted y	Predicted z $(= dy/dx)$	Correct y	Correct z
0.0	1.00000	−1.25643	1.00000	−1.25637
0.2	0.78559	−0.90162	0.78563	−0.90160
0.4	0.63447	−0.61944	0.63453	−0.61945
0.6	0.53446	−0.38716	0.53453	−0.38718
0.8	0.47750	−0.18608	0.47758	−0.18610
1.0	0.45901	0.00000	0.45910	0.00000

We use the method of Sect. 9.2.4 with $z(0) = 0$ as an initial shot and see how close we get to the target. This leads to $z(1) = 2.73672$, which is too high. Lowering our sights, we try $z(0) = -2$, which leads to -1.61962 (too low). Rather than carry on in a haphazard manner, alternately raising and lowering $z(0)$ until we hit $z(1) = 0$ we take a secant approximation regarding $z(1)$ as a function of $z(0)$. Using the formula from Table 3.6 we have as our next estimate

$$-2 + 1.61962\left(\frac{2}{2.73672 + 1.61962}\right) = -1.25643.$$

Using z(0) = −1.2564 yields the solution z(1) = 0. In all cases a step length of 0.0001 was used. Table 9.8 shows the solution at the required points.

In this case just one application of the secant formula has produced a solution that is generally correct to three decimal places. However more complicated equations may require several applications.

Discussion

The shooting method works well if there is just one unknown initial value, but its extension to equations where a number of initial values have to be determined is more problematical. In the same way that the shooting method may be implemented as finding the zero of a function of one variable, so too a problem involving finding several initial values could be regarded as finding a zero of a function of several variables. An alternative method, which approximates the differential equation by a system of linear equations, follows.

9.3.2 Difference Equations

In Chap. 6 we produced a number of formulae for approximating the derivatives of a function in terms of function values. We can make use of these formulae to reduce a differential equation to a set of approximations defined at points in the range of interest. The boundary conditions will be contained in the equations at each end of the range. If the boundary-value problem is *linear*, that is if there are no terms involving products of derivatives, we have a system of linear equations. We may solve the linear system by Gaussian elimination and so solve the differential equation in one fell swoop. By way of an example we use the previous cooling fin problem which is an example of a *linear* boundary-value problem.

Problem

Solve

$$\frac{d^2y}{dx^2} = 2y \tag{9.16}$$

over the interval [0, 1], where $y(0) = 1$ and $\frac{dy}{dx}(1) = 0$, using difference equations.

Solution

Initially we take a step-length of 0.1 to divide the x interval $[0, 1]$ into 10 equal sub-intervals. Labelling the points $x_i, i = 0, \ldots, 10$ and using the approximation to the second derivative, ((6.15), Chap. 6) we have

$$y''(x) = \frac{y(x+h) - 2y(x) + y(x-h)}{h^2}.$$

In this example $y'' = 2y$, $y_0 = 1$ and so

$$2y_i = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \quad i = 1, \ldots, 9, \quad y_0 = 1.$$

The remaining equation of the ten we need to be able to find the 10 unknowns y_i, is supplied by substituting $h = -0.1$ and the boundary condition $y'(1) = 0$ in the truncated Taylor series (6.2) to give

$$y_9 = y_{10} + h^2 y_{10}.$$

Putting $h = 0.1$ we have:

$$\begin{aligned}
-2.02y_1 \quad + y_2 \qquad\qquad\qquad\qquad &= -1 \\
y_1 - 2.02y_2 \quad + y_3 \qquad\qquad\quad &= 0 \\
y_2 - 2.02y_3 \quad + y_4 \qquad &= 0 \\
\ddots \qquad \ddots \qquad & \\
y_8 - 2.02y_9 \quad + y_{10} &= 0 \\
y_9 - 1.01y_{10} &= 0.
\end{aligned}$$

A similar system, though one involving more equations may be constructed for shorter step-lengths. The results from solving such systems are shown in Table 9.9.

The results show that we are converging to a solution and this may be confirmed by comparing with the analytic solution given in the previous problem.

Table 9.9 Boundary value problem, solution using difference equations

x	$y(x)$ solutions to $y''(x) = 2y$, various step-lengths			
	0.1	0.05	0.025	0.0125
0.1	0.8841	0.8840	0.8840	0.8840
0.2	0.7859	0.7857	0.7856	0.7856
0.3	0.7033	0.7031	0.7031	0.7030
0.4	0.6349	0.6346	0.6346	0.6345
0.5	0.5791	0.5788	0.5788	0.5787
0.6	0.5350	0.5346	0.5346	0.5345
0.7	0.5015	0.5012	0.5011	0.5010
0.8	0.4781	0.4777	0.4776	0.4776
0.9	0.4642	0.4638	0.4637	0.4637
1.0	0.4596	0.4592	0.4591	0.4591

Discussion

The system of linear equations arising from using difference equations to approximate a systems of linear differential equations may generally be written as a *banded* linear system, a system in which the non-zero elements are clustered around the main diagonal of the system matrix. As the step-length decreases the number of equations increases and such systems become *sparse*. As we have seen in Chap. 2 more economical methods than Gaussian elimination are available for their solution. However in this particular example Gaussian elimination does not cause any problems. More complicated boundary-value problems may be solved using similar approximations to the derivatives. However, if the resulting system of equations is nonlinear, iterative methods of the type described in Chap. 3 would be used to find a solution.

Summary In this chapter we have considered numerical methods for solving ordinary differential equations. You will have seen that there are two classes of problem, the initial-value problem and the boundary-value problem. Within the context of the initial-value problem we established methods for single first-order differential equations. We showed that higher order equations can be reduced to systems of first-order equations which can be solved by extending the methods already established for a single equation.

For the initial-value problem we considered the following methods and concluded that:

- Euler is stable in the sense that errors grow linearly from step to step, but may need a large number of steps.
- Runge–Kutta is more accurate than Euler and as such is generally preferred.

For the boundary-value problem we considered reducing the problem to a series of initial-value problems by means of the shooting method. We also used difference equations to discretise the whole interval. We concluded that:

- Shooting methods work well if just one initial condition is unknown.
- Difference equations are preferred if more than one initial condition is unknown.

Exercises

1. Write a Matlab program to implement Euler's method to verify the results quoted in Sect. 9.2.1.
2. Write a Matlab function *RK*2 with heading

$$ynext = function\ RK2(fn, x, y, h)$$

to advance the solution to the first-order differential equation $dy/dx = fn(x, y)$ from a solution at (x, y) to a solution at $(x = x + h)$ using the simple (second-order) Runge–Kutta method.

Assuming that the function $fn(x, y)$ has been defined in a Matlab-M file $fn.m$ the function would be called from a main program using a command such as $y2 = RK2(@fn, x1, y1, h)$.

Use the function to verify the results quoted in the chapter for

$$\frac{dy}{dx} = 3x - y + 8, \qquad y(0) = 3.$$

3. (i) Create a function *RK*4 with similar heading to *RK*2 to implement the fourth-order Runge–Kutta method. Test *RK*4 using the results quoted for the equation shown above.

(ii) Under certain conditions, taking into account gravity and air resistance, the speed v at time t of a projectile fired vertically into the air may be modelled by a first-order differential equation of the form

$$\frac{dv}{dt} = -9.81 - 0.2v^2$$

by taking into account the force due to gravity and air resistance. If the projectile is fired upwards with speed $v = 10$ show that the fourth-order Runge–Kutta method with a step length of 0.5 produces the results shown in Table 9.10. Though possible, extending the range much beyond $t = 0.6$ would not be meaningful since the projectile is about to reach zero velocity, at which point it falls to earth and a different model applies. Repeat the calculation with step-length 0.01 to suggest we have accuracy to 2 decimal places.

4. Use the function *RK*4 to solve the following simultaneous equations

$$\frac{dw}{dx} = \sin x + y$$
$$\frac{dy}{dx} = -w + \cos x$$

using the method described in Sect. 9.2.4. Advance the solutions for w and y simultaneously using commands of the form

```
w = RK4(@fxy, x, y, h);
y  = RK4(@fwx, x , w, h);
```

Table 9.10 Exercise 3, results

t	v
0	10
0.1	7.50
0.2	5.66
0.3	4.19
0.4	2.96
0.5	1.86
0.6	0.84

Table 9.11 Exercise 4, results

x	w	y
0.1	0.00517	0.09500
0.2	0.03078	0.18855
0.3	0.07654	0.27777
0.4	0.14171	0.35981
0.5	0.22520	0.43195

where $fxy = f(x, y)$ and $fwx = f(x, w)$ are functions to evaluate dw/dx and dy/dx respectively and are defined in their separate Matlab-M files, fxy.m and fwx.m.

Using $h = 0.1$ as an initial estimate for the required step-length, the results are obtained in Table 9.11. Reduce the step-lengths by a factor of 10 until the results correspond to those shown in Sect. 9.2.4.

5. Among many functions for solving differential equations Matlab provides function *ode*45 for solving systems of first-order (non-stiff)[2] differential equations of the form

$$\begin{aligned}\frac{dy_1}{dx} &= f_1(x, y_1, y_2, \ldots, y_n)\\ \frac{dy_2}{dx} &= f_2(x, y_1, y_2, \ldots, y_n)\\ \vdots &= \vdots\\ \frac{dy_n}{dx} &= f_n(x, y_1, y_2, \ldots, y_n).\end{aligned}$$

Taking the default settings a command to *ode*45 has the form

$$[X, Y] = ode45(functions, range, initialvalues)$$

[2]Differential equations which are ill-conditioned in that solutions are highly sensitive to small changes in the coefficients or boundary conditions are known as stiff equations and require special consideration.

where *range* is a vector holding the range of points of the range of x values for which a solution is required (typically [a b]) or a set of specific points ([a : increment : b]) and *initialvalues* is an n-element vector containing initial values $y_1, \ldots, y_n$ of the dependent variables. The parameter *functions* is the name of the function, which has two parameters x and y. For any value of x and a list of corresponding values $y_1, \ldots, y_n$ stored as elements of the n-element vector y *functions* returns an n-element vector of derivatives dy_i/dx.

Use *ode*45 to solve the equations of question 4 and so verify the results quoted in Sect. 9.2.4. Identifying x as the independent vector and w and y as dependent vectors a suitable command could be

$$[x, wyvec] = ode45(@sincos, [0 :\ 0.1 :\ 0.5], [0\ \ 0]);$$

where *sincos* is the name of the M-file containing

```
function dy = sincos(x, y)
  dy = [ sin(x) + y(2);  −y(1) + cos(x)];
```

The function returns w and y values (in that order) for corresponding x values in the 2-column vector *wyvec*.

6. Verify the results shown for the problem discussed in Sect. 9.3.1 using either the Matlab provided function *ode*45 or your own *RK*4 with step-length 0.00001.
7. Write a Matlab program to verify the results quoted for the example of the use of difference equations quoted in Sect. 9.3.2. This will be mainly an exercise in matrix manipulation. The program should be written to handle an arbitrary step-length h, which will determine the size of a coefficient matrix, **A** and a column vector, **b** of right-sides. Use Matlab functions *ones*, *diag*, *zeros* and the back-slash operator as shown in the program segment below, which would be appropriate for building the coefficient matrix given n equations and step-length h.

```
% column vector of n 1's
  v = ones(n, 1);
% row vector of n−1 1's
  w = ones(1, n−1);
% place −2 + h∧2 on the main diagonal
% place 1's on the diagonals above and below the main diagonal
  A = diag(w, −1) + diag( −2∗(1+h∧2)∗v, 0) + diag(w, 1);
% build the right hand side column vector
  b = zeros(n, 1);
% modify A and b to allow for the boundary conditions
  A(n, n) = −(1+h∧2);
  b(1) = −1;
% solve the equations
  x = A\b;
```

Chapter 10
Eigenvalues and Eigenvectors

Aims In this chapter we investigate methods for determining some, or all, of the eigenvalues and eigenvectors[1] of a square matrix **A**.

Overview It is convenient to divide methods for determining eigenvalues and eigenvectors into those which either

- locate a single eigenvalue/eigenvector pair

or

- determine all eigenvalues and eigenvectors.

We show how eigenvalues may be found using the characteristic polynomial and how the power method locates a single eigenvalue/eigenvector pair. We also show how eigenvalues can be found by transforming a general matrix into upper triangular form. Having shown how an eigenvector corresponding to a given eigenvalue may be found we are in a position to find all the eigenvalues and eigenvectors of a matrix.

Acquired Skills After reading this chapter you will

- understand the significance of eigenvalues and eigenvectors.
- know how to use methods for locating a single eigenvalue/eigenvector pair.
- know how to use a method for locating all eigenvalue/eigenvector pairs.
- have an appreciation of what is known in the literature as *The Eigenvalue problem*.

10.1 Introduction

Eigenvalues and eigenvectors occur in the study of differential equations which are used in modelling vibrational problems in science and engineering. Examples of

[1]The word eigen translates from the German as own, intrinsic or inherent.

DOI 10.1007/978-94-007-1366-6_10, © Springer Science+Business Media B.V. 2012

such problems include the vibration of everything from springs and simple pendulums to bridges and aeroplanes, and from the vibration of musical instruments to those of radio waves. It transpires that the solutions to such differential equations may be expressed as combinations of basic solutions each of which corresponds to a *natural vibration* of the system. Each basic solution corresponds to a particular eigenvalue of a matrix associated with the differential equation, and this connection is exploited in finding the general solution.

The natural vibrations of a structure are the vibrations at which the structure resonates and as a result may self destruct. Soldiers do not march in time when crossing a bridge for fear of accidentally marching in time with its natural vibration and so amplifying the effect and bringing down the bridge. Opera singers can achieve a similar effect by singing in the vicinity of wine glasses. It is possible to split a piece of paper by blowing on it and causing it to vibrate at a certain level. Design engineers must ensure that under normal usage vibrations at levels close to the natural vibrations of the structure, whether it be an aircraft wing, a suspension bridge or a machine component, do not occur.

Given an $n \times n$ matrix, $\mathbf{A}$ we aim to find one or more values for λ (an eigenvalue) and corresponding vector $\mathbf{x}$ (an eigenvector) defined by the relation

$$\mathbf{Ax} = \lambda\mathbf{x}.$$

Problem

Verify that $\mathbf{x} = (1, 2, 1)$ is an eigenvector of the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix}$$

with corresponding eigenvalue 3.

Solution

Multiplying $\mathbf{x}$ by $\mathbf{A}$ and comparing with $3\mathbf{x}$ gives the required result.

We have

$$\mathbf{Ax} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \\ 3 \end{pmatrix} = 3 \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}.$$

Problem

Suppose that 3 is known to be an eigenvalue of the matrix, $\mathbf{A}$ of the previous problem. Find the corresponding eigenvector.

Solution

We solve the system of equations $\mathbf{Ax} = 3\mathbf{x}$. Letting $\mathbf{x} = (x_1, x_2, x_3)$ we have

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 3x_1 \\ 4x_1 - 3x_2 + 8x_3 &= 3x_2 \\ x_1 + 2x_3 &= 3x_3 \end{aligned}$$

and so

$$\begin{aligned} -x_1 + x_2 - x_3 &= 0 \\ 4x_1 - 6x_2 + 8x_3 &= 0 \\ x_1 - x_3 &= 0. \end{aligned}$$

Gaussian elimination produces the upper triangular system

$$\begin{aligned} -x_1 + x_2 - x_3 &= 0 \\ -2x_2 + 4x_3 &= 0 \\ 0x_3 &= 0. \end{aligned}$$

Thus there is no unique solution and so we are free to set one of the variables to whatever we choose. Letting $x_3 = 1$, we have $x_2 = 2$ and $x_1 = 1$ from back-substitution.

Discussion

We have shown in this example that eigenvectors are not unique. By setting x_3 to *any* constant we can show that any multiple of $(1, 2, 1)$ is an eigenvector of $\mathbf{A}$ corresponding to the eigenvalue 3.

For any vector $\mathbf{v}$ the **normalised form** is defined to be $\mathbf{v}/norm(\mathbf{v})$ and so the norm of a normalised vector is 1. Being able to assume that vectors (and in particular eigenvectors) are presented in normalised form can sometimes simplify subsequent discussion.

10.2 The Characteristic Polynomial

We may write $\mathbf{Ax} = \lambda\mathbf{x}$ in the form

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = 0$$

where, as before, $\mathbf{x}$ is the eigenvector corresponding to an eigenvalue λ of the matrix $\mathbf{A}$ of order n and $\mathbf{I}$ is the identity matrix of order n. The determinant of the matrix $\mathbf{A} - \lambda\mathbf{I}$ is a polynomial in λ and is known as the **characteristic polynomial**. The characteristic polynomial provides a means of identifying eigenvalues.

Problem

Use the characteristic polynomial to find the eigenvalues of the matrix

$$\mathbf{A} = \begin{pmatrix} 4 & 1 \\ -5 & -2 \end{pmatrix}.$$

Solution

The characteristic polynomial, $p(\lambda)$ is given by the determinant of the matrix

$$\begin{pmatrix} 4-\lambda & 1 \\ -5 & -2-\lambda \end{pmatrix}.$$

Therefore

$$\begin{aligned} p(\lambda) &= (4-\lambda)(-2-\lambda)+5 \\ &= \lambda^2 - 2\lambda - 3 \\ &= (\lambda-3)(\lambda+1). \end{aligned}$$

Since the system of equations, $\mathbf{Ax} = \lambda\mathbf{x}$ does not have a unique solution, the coefficient matrix, $\mathbf{A}$ is singular, and so columns (and rows) are linearly dependent which is equivalent to the determinant being zero. Setting $p(\lambda) = 0$ we have eigenvalues, $\lambda = 3$ and $\lambda = -1$.

Discussion

In principle we have shown a method for locating eigenvalues, although the cost of forming p and then using an appropriate root-finding technique from Chap. 3 might be prohibitively time consuming for large matrices. The characteristic polynomial can be used to determine all the eigenvalues but information as to eigenvectors would have to be found by solving the appropriate linear system. Note that the characteristic polynomial indicates that an $n \times n$ matrix has n eigenvalues and so it is possible that some eigenvalues may be identical and some may occur as complex conjugate pairs.

10.3 The Power Method

We present a method that can be used to find the largest eigenvalue of a matrix and the corresponding eigenvector. Essentially the same method may be applied to find the smallest eigenvalue and corresponding eigenvector. The method is explained using the example below. We use the terms *largest* and *smallest* to mean largest and smallest in absolute value.

Table 10.1 Power method, largest eigenvalue

$\mathbf{x}_i$			η_{i+1}
1.0000	1.0000	1.0000	9.0000
0.2222	1.0000	0.3333	1.1111
1.0000	0.500	0.8000	8.9000
0.1910	1.0000	0.2921	1.08999
⋮			⋮
−0.1775	1.000	0.0326	−3.4495
−0.1775	1.000	0.0326	−3.4495

Problem

Find the largest eigenvalue of the matrix,

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix}.$$

Solution

We look for the largest λ and corresponding $\mathbf{x}$ to satisfy

$$\mathbf{Ax} = \lambda\mathbf{x}.$$

We approach a solution in an iterative manner by producing a sequence of vectors $\mathbf{x}_i$, $i = 1, 2, \ldots$ using the formula

$$\begin{aligned} \eta_{i+1} &= \mathit{Largest}(\mathbf{Ax}_i) \\ \mathbf{x}_{i+1} &= \frac{1}{\eta_{i+1}}\mathbf{Ax}_i, \quad i = 1, 2, \ldots, \end{aligned} \tag{10.1}$$

where $\mathbf{x}_1$ is any non-zero vector and $\mathit{Largest}(x)$ denotes the element of largest absolute value of a vector x. In the absence of any other information we start with $\mathbf{x}_1 = (1, 1, 1)$. The method is slow to converge but eventually we have the results in Table 10.1. We conclude that to the accuracy shown the largest eigenvalue is 3.4495 and that the corresponding eigenvector (in unnormalised form) is $(-0.1775\ \ 1.0000\ \ 0.0326)$. This result may be confirmed by finding the roots of the characteristic equation (Exercise 1). Theoretical justification for the general case follows.

10.3.1 Power Method, Theory

We show that the eigenvectors of a matrix corresponding to distinct eigenvalues are linearly independent, a result that will be needed later. The proof is by contradiction.

We assume the eigenvectors of the matrix are linearly dependent and show that this leads to a contradiction.

Suppose that the $n \times n$ matrix $\mathbf{A}$ has distinct eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_m$ with corresponding eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m$. We know from the degree of the characteristic polynomial that m must be $\leq n$. If $\mathbf{x}_k$ is the *first* of the eigenvectors to be a linear combination of the eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k-1}$ we have

$$\mathbf{x}_k = \sum_{i=1}^{k-1} \alpha_i \mathbf{x}_i \tag{10.2}$$

where not all the α_i are zero. By pre-multiplying both sides of (10.2) by $\mathbf{A}$ we find that this implies

$$\mathbf{A}\mathbf{x}_k = \sum_{i=1}^{k-1} \alpha_i (\lambda_k - \lambda_i)\mathbf{x}_i = 0. \tag{10.3}$$

Since the terms $\lambda_k - \lambda_i$ in the above are not all zero (10.3) shows that $\mathbf{x}_k$ is not the first of the eigenvectors to be a linear combination of the eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{k-1}$. This contradiction provides the required result, which we use to establish the validity of (10.1).

We assume that we have an $n \times n$ matrix $\mathbf{A}$ with distinct n eigenvalues, which as we have seen will have corresponding eigenvectors ξ_i, $i = 1, \ldots, n$ that are linearly independent. We also assume that the eigenvectors and eigenvalues λ_i are ordered so that

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|.$$

Since the eigenvectors of $\mathbf{A}$ are linearly independent for any vector $\mathbf{x}$ of the same dimension we have

$$\mathbf{x} = \sum_{i=1}^{n} \alpha_i \xi_i$$

where the α_i are constants and the ξ_i are the eigenvectors of $\mathbf{A}$ with corresponding eigenvalues λ_i. We make the further assumptions that $\alpha_1 \neq 0$ and that the eigenvalues are real (not complex). It may appear that we are making a number of assumptions but the procedure may be modified to deal with more general situations. It follows that

$$\mathbf{A}\mathbf{x} = \sum_{i=1}^{n} \mathbf{A}\alpha_i \xi_i = \sum_{i=1}^{n} \alpha_i \lambda_i \xi_i$$

and so

$$\mathbf{A}^k\mathbf{x} = \sum_{i=1}^{n} \alpha_i \lambda_i^k \xi_i = \lambda_1^k \sum_{i=1}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k \xi_i.$$

Going to the limit and noting that ξ_1 is the eigenvalue of largest absolute value

$$\lim_{k\to\infty} \mathbf{A}^k\mathbf{x} = \lim_{k\to\infty} \alpha_1 \lambda_1^k \xi_1$$

and so taking largest values

$$\lim_{k\to\infty} \frac{Largest(\mathbf{A}^{k+1}\mathbf{x})}{Largest(\mathbf{A}^{k}\mathbf{x})} = \lim_{k\to\infty} \frac{\lambda_1^{k+1}}{\lambda_1^{k}}$$
$$= \lambda_1 .$$

We can connect this results with the sequence (10.1), which generates a sequence η_i

$$\eta_3 = \frac{Largest(\mathbf{A}^2\mathbf{x}_1)}{Largest(\mathbf{A}\mathbf{x}_1)}, \quad \eta_4 = \frac{Largest(\mathbf{A}^3\mathbf{x}_1)}{Largest(\mathbf{A}^2\mathbf{x}_1)}, \quad \ldots, \quad \eta_{k+1} = \frac{Largest(\mathbf{A}^k\mathbf{x}_1)}{Largest(\mathbf{A}^{k-1}\mathbf{x}_1)}$$

and confirms that $\eta \to \lambda_1$, whatever the value of $\mathbf{x}_1$.

We may use the same technique to find the smallest eigenvalue of a matrix. For any eigenvalue λ and corresponding eigenvector $\mathbf{x}$ of a matrix $\mathbf{A}$ we have

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad \text{and so} \quad \frac{1}{\lambda}\mathbf{x} = \mathbf{A}^{-1}\mathbf{x}.$$

In particular, if λ is the smallest eigenvalue of $\mathbf{A}$, then it is the largest eigenvalue of the inverse, $\mathbf{A}^{-1}$. The corresponding eigenvector is the same in each case. If we apply the power method to finding the largest eigenvalue of $\mathbf{A}^{-1}$ we would approach a solution in an iterative manner by producing a sequence of vectors $\mathbf{x}_i$, $i = 1, 2, \ldots$ using the formula

$$\mathbf{A}^{-1}\mathbf{x}_i = \lambda_i \mathbf{x}_{i+1}, \quad i = 1, 2, \ldots. \tag{10.4}$$

However, rather than calculate $\mathbf{A}^{-1}$ directly, we use the following form.

$$\lambda_i \mathbf{A}\mathbf{x}_{i+1} = \mathbf{x}_i, \quad i = 1, 2, \ldots. \tag{10.5}$$

Starting with an arbitrary $\mathbf{x}_1$ at each step we would solve the linear system

$$\mathbf{A}\mathbf{x}_{i+1} = \frac{1}{\lambda_i}\mathbf{x}_i, \quad i = 1, 2, \ldots$$

where λ_i is the element of largest absolute value of the vector x_i. The method is illustrated in the following problem.

Problem

Find the smallest eigenvalue of the matrix $\mathbf{A}$ given by

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix}.$$

Solution

Starting with $\mathbf{x}_1 = (1, 1, 1)$ and setting λ_i to the value of the maximum element of $\mathbf{x}_i$, we solve the linear system (10.5) at each step to find $\mathbf{x}_{i+1}$ to give the results shown in Table 10.2. However if the linear system is solved at the first step

Table 10.2 Power method, smallest eigenvalue

$\mathbf{x}_i$			λ_{i+1}
1.0000	1.0000	1.0000	1.0000
0.2000	1.0000	0.4000	1.0000
0.0800	0.2000	0.1600	0.2000
0.0267	0.7333	0.3867	0.7333
⋮			⋮
−0.2914	0.6898	0.5294	0.6899
−0.2914	0.6899	0.5294	0.6899

by Gaussian Elimination we can avoid unnecessary repetition by saving the **LU** decomposition (page 24) of **A** for use in subsequent steps. We conclude that the largest eigenvalue of $\mathbf{A}^{-1}$ is 0.6899 and so the smallest eigenvalue of **A** is $1/0.6899$ namely 1.4495. The corresponding unnormalised eigenvector is (−0.2914 0.6899 0.5294) or (−0.3178 0.7522 0.5773) in normalised form.

Discussion

A similar method may be used to find the eigenvalue of a matrix closest to some given number. For example to find the eigenvalue of a matrix **A** closest to 10 we would look for the smallest eigenvalue of $\mathbf{A} - 10\mathbf{I}$, where **I** is the identity matrix and provided that **A** satisfies conditions already stated for the power method. For any eigenvalue, λ of $\mathbf{A} - 10\mathbf{I}$ it follows that $\lambda + 10$ is an eigenvalue of **A**.

This completes the discussion of methods for finding individual eigenvalues. We could find all the eigenvalues of a given matrix by fishing around to find the eigenvalues nearest to a succession of estimates until all the eigenvalues had been isolated. However as an alternative to such a haphazard approach we consider a direct method for finding all eigenvalues simultaneously.

10.4 Eigenvalues of Special Matrices

We point the way to establishing a method to find all the eigenvalues and eigenvectors of a general matrix by considering two special cases.

10.4.1 Eigenvalues, Diagonal Matrix

Consider the matrix **A** given by

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 7 \end{pmatrix}.$$

The characteristic polynomial, $p(\lambda)$ for $\mathbf{A}$ is given by

$$p(\lambda) = (\lambda - 2)(\lambda + 3)(\lambda - 7).$$

The roots of $p(\lambda)$ are clearly 2, -3 and 7 and so we have the eigenvalues. The corresponding eigenvectors are $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$.

The result generalises. The eigenvalues of an $n \times n$ diagonal matrix are the diagonal elements. The corresponding eigenvectors are the columns of the identity matrix of order n namely, $(1, 0, \dots, 0), (0, 1, 0, \dots, 0), \dots, (0, 0, \dots, 0, 1)$.

10.4.2 Eigenvalues, Upper Triangular Matrix

Consider the matrix $\mathbf{A}$ given by

$$\mathbf{A} = \begin{pmatrix} 2 & 2 & 1 \\ 0 & -3 & 4 \\ 0 & 0 & 7 \end{pmatrix}.$$

The characteristic polynomial, $p(\lambda)$ for $\mathbf{A}$ is given by the determinant

$$p(\lambda) = \begin{vmatrix} 2-\lambda & 2 & 1 \\ 0 & -3-\lambda & 4 \\ 0 & 0 & 7-\lambda \end{vmatrix}$$

and so the eigenvalues are the diagonal elements, 2, -3 and 7, as for a diagonal matrix. The result generalises to upper triangular matrices of any order.

10.5 A Simple QR Method

Since we have a method for finding the eigenvalues (and hence the eigenvectors) of upper triangular matrices, we aim to transform more general matrices into this form in such a way that we do not lose track of the original eigenvalues. If λ is an eigenvalue of $\mathbf{A}$ with corresponding eigenvector $\mathbf{x}$ we have for any matrix $\mathbf{R}$

$$\mathbf{RAx} = \lambda \mathbf{Rx}. \tag{10.6}$$

Furthermore

$$\left(\mathbf{RAR}^{-1}\right)\mathbf{Rx} = \lambda \mathbf{Rx}$$

and so $\mathbf{RAR}^{-1}$ has the same eigenvalues as $\mathbf{A}$ but with corresponding eigenvectors $\mathbf{Rx}$, where $\mathbf{x}$ is an eigenvector of $\mathbf{A}$. Such a transformation of $\mathbf{A}$ to $\mathbf{RAR}^{-1}$ is known as a **similarity transformation**. If we can make a similarity transformation of $\mathbf{A}$ to upper triangular form, the eigenvalues of the upper triangular matrix, namely the values of the diagonal elements, will be the eigenvalues of $\mathbf{A}$. The eigenvectors of $\mathbf{A}$ can be recovered from those of $\mathbf{RAR}^{-1}$.

Problem

Find the eigenvalues of the matrix **A** given by

$$\mathbf{A} = \begin{pmatrix} 4 & -1 & 2 & 1 \\ 1 & 1 & 6 & 3 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix} \tag{10.7}$$

by making a similarity transformation to upper triangular form.

Solution

But for the presence of coefficient 1 at position (2, 1) the matrix **A** would already be in upper triangular form. Accordingly we aim for a zero coefficient at this position by pre-multiplying **A** by the matrix $\mathbf{Q}_1$ given by

$$\mathbf{Q}_1 = \begin{pmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

for suitable c and s. Multiplying, we have

$$\mathbf{Q}_1\mathbf{A} = \begin{pmatrix} 4c+s & -c+s & 0 & 0 \\ -4s+c & s+c & 0 & 0 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

and so values of the existing zeros below the diagonal are not affected. To reduce the coefficient in position (2, 1) to zero we require

$$-4s + c = 0.$$

In addition to we choose c and s to satisfy

$$c^2 + s^2 = 1.$$

Substituting for s produces the solution $c = 0.9701$, $s = 0.2425$. Since we have chosen $c^2 + s^2 = 1$ it follows that $\mathbf{Q}\mathbf{Q}^T = \mathbf{I}$ (the identity matrix) and so the inverse of $\mathbf{Q}$ is given by $\mathbf{Q}^{-1} = \mathbf{Q}^T$. Having an immediate expression for the inverse saves a lot of work.

Using

$$\mathbf{Q} = \begin{pmatrix} 0.9701 & 0.2425 & 0 & 0 \\ -0.2425 & 0.9701 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

we have

$$\mathbf{Q}_1\mathbf{A}\mathbf{Q}_1^{-1} = \begin{pmatrix} 3.8235 & -1.7059 & 3.3955 & 1.6977 \\ 0.2941 & 1.1765 & 5.3358 & 2.6679 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix}.$$

This is not upper triangular. If it was we would have solved the problem, but it is closer to being upper triangular than $\mathbf{A}$ so we repeat the procedure on $\mathbf{Q}_1\mathbf{A}\mathbf{Q}_1^{-1}$ which we note still has the same eigenvalues as $\mathbf{A}$.

Repeating the process again (and again), recalculating the constants c and s we obtain a sequence of matrices

$$\mathbf{Q}_2\mathbf{Q}_1\mathbf{A}\mathbf{Q}_1^{-1}\mathbf{Q}_2^{-1} = \begin{pmatrix} 3.7000 & -1.9000 & 3.7947 & 1.8974 \\ 0.1000 & 1.3000 & 5.0596 & 2.5298 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

and

$$\mathbf{Q}_3\mathbf{Q}_2\mathbf{Q}_1\mathbf{A}\mathbf{Q}_1^{-1}\mathbf{Q}_2^{-1}\mathbf{Q}_3^{-1} = \begin{pmatrix} 3.6496 & -1.9635 & 3.9300 & 1.9650 \\ 0.0365 & 1.3504 & 4.9553 & 2.4776 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix}$$

and so on, eventually leading to

$$\begin{pmatrix} 3.6180 & -2.0000 & 4.0092 & 2.0046 \\ 0.0000 & 1.3820 & 4.8914 & 2.4457 \\ 0 & 0 & -3 & 4 \\ 0 & 0 & 0 & 7 \end{pmatrix}.$$

Reading the diagonal values it follows that within the limits of the computation the eigenvalues of the original matrix $\mathbf{A}$ are 3.6180, 1.3820, -3 and 7.

In this problem it was only necessary to reduce one element to zero. In the next problem look to extend the method to the general case of more than one non-zero below the diagonal.

Problem

Find the eigenvalues of the matrix $\mathbf{A}$ given by

$$\mathbf{A} = \begin{pmatrix} 3 & 5 & -4 & 4 \\ -1 & -3 & 1 & -4 \\ 4 & -2 & 3 & 5 \\ 3 & -5 & 5 & 4 \end{pmatrix}.$$

Solution

In the previous problem it was necessary to reduce one element to zero namely, the element at position (2, 1) in the matrix. We did this by pre-multiplying $\mathbf{A}$ by a matrix $\mathbf{Q}$ formed from the unit diagonal matrix on which a 2×2 matrix $\left(\begin{smallmatrix} c & s \\ -s & c \end{smallmatrix}\right)$ with appropriate values for c and s was superimposed with bottom left-hand corner at position (2, 1). The method was repeated until we found a matrix $\mathbf{Q}$ such that $\mathbf{QAQ}^{-1}$ was upper-triangular.

The position of the 2×2 matrix was chosen so that the pre-multiplying would not change any of the elements below the diagonal that were already zero. In this problem we must be careful to do the same. We work up the columns below the diagonal of **A** reducing elements to zero, one by one at positions (4, 1), (3, 1), (2, 1), (4, 2), (3, 2) and (4, 3) by superimposing the 2×2 matrix with appropriate c and s with the bottom left-corner at positions (4, 3), (3, 2), (2, 1), (4, 3), (3, 2) and (4, 3). A pattern for the general case may be inferred. As before the aim is to produce a similarity transformation of **A** to upper-triangular form, from which we may read the eigenvalues.

On first iteration we have the sequence

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.8 & 0.6 \\ 0 & 0 & -0.6 & 0.8 \end{pmatrix} \begin{pmatrix} 3 & 5 & -4 & 4 \\ -1 & -3 & 1 & -4 \\ 4 & -2 & 3 & 5 \\ 3 & -5 & 5 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 5 & -4 & 4 \\ -1 & -3 & 1 & -4 \\ 5 & -4.6 & 5.4 & 6.4 \\ 0 & -2.8 & 2.2 & 0.2 \end{pmatrix}$$

and then

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -0.1961 & 0.9806 & 0 \\ 0 & -0.9806 & -0.1961 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 5 & -4 & 4 \\ -1 & -3 & 1 & -4 \\ 5 & -4.6 & 5.4 & 6.4 \\ 0 & -2.8 & 2.2 & 0.2 \end{pmatrix}$$
$$= \begin{pmatrix} 3 & 5 & -4 & 4 \\ 5.0990 & -3.9223 & 5.0990 & 7.0602 \\ 0 & 3.8439 & -2.0396 & 2.6672 \\ 0 & -2.8 & 2.2 & 0.2 \end{pmatrix}$$

and so on until all elements below the diagonal to reduce to zero and we have a sequence

$$\mathbf{Q}_6\mathbf{Q}_5 \cdots \mathbf{Q}_1\mathbf{A} = \begin{pmatrix} 5.9161 & -0.8452 & 2.3664 & 8.1135 \\ 0 & -7.8921 & 6.5888 & -1.1223 \\ 0 & 0 & -1.4097 & -2.2661 \\ 0 & 0 & 0 & -0.8812 \end{pmatrix}.$$

However

$$\mathbf{Q}_6\mathbf{Q}_5 \cdots \mathbf{Q}_1\mathbf{A}\mathbf{Q}_1^{-1}\mathbf{Q}_2^{-1} \cdots \mathbf{Q}_6^{-1} = \begin{pmatrix} 8.8571 & 0.7221 & 1.8162 & -4.9874 \\ 5.2198 & -2.6003 & -7.8341 & 3.4025 \\ -2.1022 & -1.5678 & 0.1808 & 0.4609 \\ -0.4469 & -0.5104 & -0.0103 & 0.5623 \end{pmatrix}$$

is not upper triangular though it appears to be a step in the right direction. We repeat the procedure taking this matrix as the next to be transformed.

Starting with the original matrix **A**, the matrices at the end of each step of the process eventually converge to upper triangular form. The diagonal elements of which are shown in Table 10.3. We conclude that the eigenvalues of **A** are 9.1390, −4.8001, 2 and 0.6611, to the accuracy shown.

Table 10.3 Eigenvalues, simple QR method

Diagonal elements			
8.8571	−2.6003	0.1808	0.5623
9.2761	−5.0723	2.3788	0.4174
9.3774	−4.7131	1.7347	0.6011
9.0837	−4.8242	2.1016	0.6388
9.1839	−4.7996	1.9616	0.6541
9.1188	−4.7962	2.0187	0.6587
9.1504	−4.8041	1.9934	0.6603
⋮			⋮
9.1391	−4.8002	2.0000	0.6611
9.1390	−4.8001	2.0000	0.6611
9.1390	−4.8001	2.0000	0.6611

Table 10.4 Simple QR method

Step 1	Find a matrix $\mathbf{Q}$ such that $\mathbf{QA}$ is upper triangular
Step 2	If $\mathbf{QAQ}^{-1}$ is upper triangular then stop. The elements of the diagonal are the eigenvalues
Step 3	Repeat from step 1

Discussion

The method we have described for finding the eigenvalues of a general matrix may be formalised as shown in Table 10.4. The QR method was invented by Francis[2] in 1961 who used the terms Q and R, hence the name. The version we have described is a simplified version, which though theoretically sound can be slow to converge and may be defeated by rounding error. Problems involving matrices having equal or complex eigenvalues have not been considered. Over time a great deal of work has been done to improve QR, which is one of the most widely used and important methods of numerical computation.

Summary In this chapter we have considered the practical importance of *The Eigenvalue problem* in science and engineering. We have shown the need for calculating eigenvalues, which may be achieved in a number of ways. For finding single eigenvalues we described the characteristic polynomial and the power method. Having found an eigenvalue we showed how the corresponding eigenvector could be found by solving a linear system. We showed how eigenvectors may be expressed in normalised form.

[2]John G.F. Francis, English computer scientist (1934–). Working independently Russian mathematician Vera Kublanovskaya also proposed the method in the same year.

To introduce the simple QR method for finding all the eigenvalues of a matrix we considered two special cases. We showed how the eigenvalues of a diagonal matrix and an upper triangular matrix could be found from the diagonal elements. This suggested a means for obtaining the eigenvalues of a more general matrix. We effected a transformation (a similarity transformation) to upper triangular form so that original eigenvalues were preserved. We indicated that our simple QR method could be made more efficient but that it did contain the essential ingredients of the more powerful methods which are currently available.

Exercises

1. Either by writing your own function or by using the Matlab function *poly* create the coefficients of the characteristic polynomial of the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix}.$$

We have seen that the matrix has an eigenvalue 3 (page 216). Use the characteristic equation to find the remaining eigenvalues either directly or by using the Matlab function *roots*.

A suitable sequence of commands would be

```
A = [ 2 1 −1; 4 −3 8; 1 0 2];
p = poly(A); % get the coefficients of the characteristic polynomial
ev = roots(p); % store the roots (the eigenvalues) in the vector ev
```

Verify that the eigenvalues are in agreement with the results given in the chapter.

2. Find the (normalised) eigenvector corresponding to each of the eigenvalues found in question 1 by considering the solutions to $\mathbf{Ax} = \lambda\mathbf{x}$, for a given λ. Start by writing the equation in the form $\mathbf{B}x = 0$, where $\mathbf{B} = \mathbf{A} - \lambda\mathbf{I}$ and $\mathbf{I}$ is the identity matrix.

Use Matlab function *lu* to find the LU decomposition of $\mathbf{B}$ and consider the linear system $\mathbf{LUx} = 0$, which reduces to the upper triangular system $\mathbf{Ux} = 0$. Verify that for $\lambda = 5$ we have the system

$$\begin{aligned} 4x_1 - 2x_2 + 8x_3 &= 0 \\ -0.5x_2 + 5x_3 &= 0 \\ 0x_3 &= 0. \end{aligned}$$

As might be expected this is an indeterminate system (we could set x_3 to any number) since any eigenvalue corresponding to a given eigenvector is a multiple of the normalised eigenvector. In order to proceed to a solution modify the last equation to $x_3 = 1$ and proceed to a solution using the Matlab left division operator. Verify that the solution is $x = (3, 10, 1)$ and so the normalised eigenvector corresponding to eigenvalue 5 ($x/norm(x)$) is $(0.2860, 0.9535, 0.0953)$.

Verify that the method works equally well for complex eigenvalues and eigenvectors using exactly the same program.

Answers may also be verified using the Matlab supplied function *eig* for finding eigenvalues and eigenvectors. For a given matrix **A**, the command $[v, e] = eig(A)$ would produce a matrix, v whose columns are eigenvectors of A and a diagonal matrix, e of corresponding eigenvalues.

3. Write a program to implement the power method. Make two versions, one for the largest eigenvalue and another for the smallest (in absolute values). Use the programs to verify the results shown in Sect. 10.3. In addition use one of the programs to find the eigenvalue of

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & -1 \\ 4 & -3 & 8 \\ 1 & 0 & 2 \end{pmatrix}$$

closest to -0.5.

4. (i) Write a program to apply the simple QR method to the first problem of Sect. 10.5. Build the sequence $Q_1 A Q_1^{-1}, Q_2 Q_1 A Q_1^{-1} Q_2^{-1}, \ldots$ with each iteration until the absolute value of element $(2, 1)$ of the matrix QAQ^{-1} where $Q = Q_n \cdots Q_2 Q_1$, for some n is less than 1.0e-6.

Having completed the code for Step 1 of the QR method, Step 2 follows by wrapping Step 1 in a *while* loop. Assuming **A** has been established the program might take the following form

```
while abs(A(2, 1)) > 1.0e-6;
  % solve -A(1, 1)*s + A(2, 1)*c = 0; s∧ 2 + c∧2 = 1;
  s = 1/sqrt(1 + (A(1, 1)/A(2, 1))∧2);
  c = s *( A(1, 1)/A(2, 1));
  Q = [ c s 0 0; −s c 0 0; 0 0 1 0; 0 0 0 1];
  A = Q*A*Q'; % inverse Q = Q'
end;
```

(ii) Find the normalised eigenvectors corresponding to the eigenvalues found in part(i) using the method of question 1.

Note that Matlab supplies a function *qr* to provide the QR factorisation of a matrix. For example a command of the form [Q R] = qr(A); would for a given matrix **A** return a unitary **Q** (a matrix such that $\mathbf{Q}^{-1} = \mathbf{Q}^T$) and upper triangular matrix **R** such that **QR** = **A**. However when applied to the examples quoted in the chapter the function takes a slightly different route in that the signs of variables c and s (using the notation of the chapter) are reversed, in solving for c and s the negative square root is chosen.

5. Extend the program of question 4(i) to apply the QR method to the second problem of Sect. 10.5. You might find it easier to repeat similar code from the previous question, cutting and pasting with a little editing to make separate code for each of the six elements to be reduced to zero. Having spotted the pattern you might then try to shorten the program by using suitable loops to calculate c and s (using the notation of the main text) and position the c and s sub-matrix automatically. Use the norm of the vector of diagonal elements of $\mathbf{QAQ}^{-1}$ at the end of step 1 as the test for convergence.

Chapter 11
Statistics

Aims This chapter aims to

- introduce the statistical concepts which are frequently used whenever data, and the results of processing that data, are discussed.
- discuss how computers can generate data to be used for test and simulation purposes.

Overview We give formal definitions of a number of statistical terms, such as *standard deviation* and *correlation*, which are in every-day use and illustrate their relevance by way of examples. We quote a number of formulae connecting the various terms.

We reconsider the fitting of a least squares straight line to data, first discussed in Chap. 4. We show how the results from this method may be presented in such a way that we have a statistical estimate as to the accuracy of fit.

Finally, we discuss the production of random numbers. We establish the need for such values and how they might be generated. In so doing we question what we mean by a random number and suggest ways for testing how a sequence of numbers may be regarded as being a random selection. As a by-product of random number generation we propose a method for integration to add to those of Chap. 5.

Acquired Skills After reading this chapter you will

- understand what is meant by a number of statistical terms including *random variable*, *stochastic variable*, *histogram*, *frequency*, *frequency distribution*, *normal distribution*, *uniform distribution*, *expected value*, *average*, *mean*, *variance*, *standard deviation*, *unbiased statistic*, *covariance*, *correlation*, *correlation coefficient*, *confidence interval*, *random number* and *pseudo-random number*.

You will also be able to

- calculate the *mean* and *standard deviation* of a set of numbers and by calculating *correlation coefficients* be able to decide if different sets of numbers are related.
- understand the relevance of Chebyshev's theorem to the *mean* and *standard deviation* of a set of numbers.

DOI 10.1007/978-94-007-1366-6_11, © Springer Science+Business Media B.V. 2012

- supply confidence intervals to the values of coefficients resulting from the least squares straight line approximation to given data.
- generate random numbers to meet the needs of any particular application.
- use the Monte Carlo method for numerical integration.

11.1 Introduction

There is no doubt that scientists, engineers, economists, sociologists and indeed anyone called upon to interpret data in either a professional or personal capacity (a definition which encompasses practically the whole population) should have a working knowledge of statistics. Whilst not necessarily agreeing wholeheartedly with the popular sentiment that there are *lies*, *damned lies* and *statistics*, at the very least we should be equipped to appraise critically the comments of politicians and others who would seek to influence us by the use (and misuse) of statistics.

Statistics is too large a subject to be taught as part of a conventional numerical methods course and for that reason is often omitted completely. What follows only skims the surface and in order to cover a reasonable amount of ground a good deal of mathematical detail has been omitted.

11.2 Statistical Terms

11.2.1 Random Variable

Any quantity which varies from one repetition of a process to the next is said to be a *random* or *stochastic* variable. An example of a random variable is the first number to be called in a lottery draw. Another example would be the mid-temperature in central London.

11.2.2 Frequency Distribution

A diagram such as Fig. 11.1, which shows the number of students having Computing marks within a given range, is known as a *histogram*. A histogram is a series of rectangles which show the number of times (or *frequency*) that a particular event (examination mark, height, weight, temperature, or whatever) occurs. The width of each rectangle is proportional to a range of values and the height is proportional to the frequency within that range.

Alternatively we could use a graph to record a series of points to show the frequency of a particular event. The curve which is formed by linking the sequence of points is known as a *frequency distribution*. A frequency distribution having the

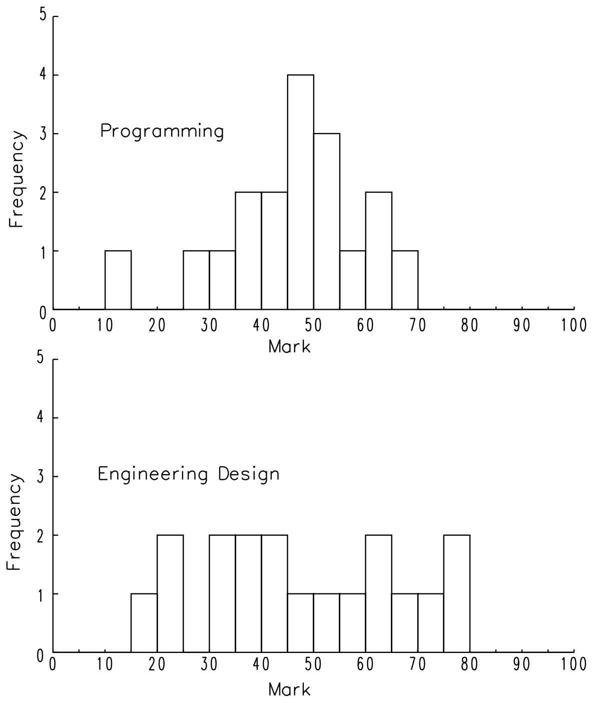


Fig. 11.1 Examination marks

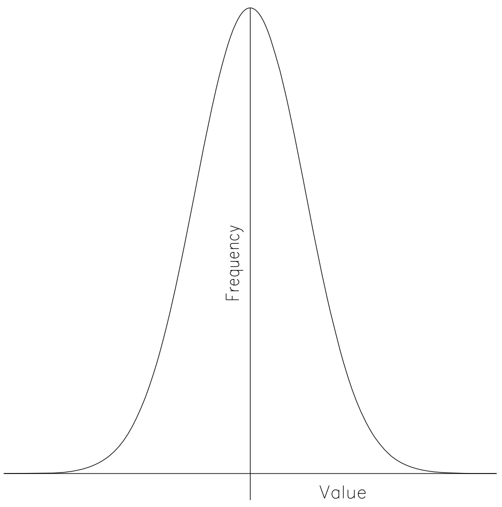


Fig. 11.2 Normal distribution

familiar well-balanced bell shape (Fig. 11.2) is known as a *normal* distribution. The normal distribution is a good model for the observed frequencies of many animal and human characteristics, psychological measurements, physical phenomena and the outcomes of economic and sociological activities and as such has a prominent role in statistics.

11.2.3 Expected Value, Average and Mean

The *expected value* of a random variable is defined to be its average value. However, in statistics the term *mean* is preferred to *average*, and so we use this term henceforth. We use $E(x)$ to denote the expected value of a random variable x. Thus if x is observed to take values $x_1, x_2, \dots, x_n$ we have

$$E(x) = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

We can extend the definition to define the expected value of a continuous function $f(x)$ over an interval $[a, b]$. We divide the interval into n equal sections $[x_i, x_{i+1}]$ $(x_1 = a, x_{n+1} = b)$ and approximate the area under the curve by the sum of the areas of the rectangles of height $f(x_i)$ and width $\frac{b-a}{n}$. This is a simpler approximation than that inherent in the trapezium rule of Chap. 5 but it serves our purpose. We have

$$\frac{b-a}{n}\sum_{i=1}^{n} f(x_i) \approx \int_a^b f(x)\,dx$$

and so in the limit as n tends to infinity

$$E\big(f(x)\big) = \frac{1}{b-a}\int_a^b f(x)\,dx. \tag{11.1}$$

11.2.4 Variance and Standard Deviation

The expected value of a random variable does not necessarily indicate how the values are dispersed. For example, a set of examination marks which are bunched around the 50% level may have the same average as marks which are more widely dispersed. Quite different conclusions might be drawn from the two sets of results. We introduce *variance* and *standard deviation* as measures of the dispersion of a set of values taken by a random variable.

If x is a single random variable having an expected value $E(x)$, the variance, $Var(x)$, of x is defined by

$$Var(x) = E\big(\big[x - E(x)\big]^2\big).$$

The definition of variance extends to a set of variables $x_1, x_2, \dots x_n$, with mean value $E(x)$ and is usually symbolised by σ^2, where

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}\big(x_i - E(x)\big)^2.$$

The standard deviation is defined to be the square root of the variance and so variance is often denoted by σ^2 and standard deviation by σ.

However standard deviation is invariably computed according to the formula

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(\left[x_i - E(x)\right]^2\right)}. \tag{11.2}$$

There is a somewhat tortuous statistical justification for using $n-1$ rather than n based on the following, which may be tested by experiment to provide a more reliable estimate. The formula is based on the values $x_1, x_2, \dots, x_n$ which are often just a sample of all possible values. For example the x_i values may result from a survey of journey times to work which would have probably been based on a sample group of employees. In this case $E(x)$ in the formula is based on just a selection of all possible x_i values and so is not an independent variable. This restriction reduces the degrees of freedom inherent in the equation from n to $n-1$.

The standard deviation indicates where the values of a frequency distribution are located in relation to the mean. According to Chebyshev's theorem at least 75% of values will fall within ± 2 times the standard deviation of the mean and at least 89% will fall within ± 3 times the standard deviation of the mean. For a normal distribution the proportions of variables falling within 1, 2 and 3 standard deviations are respectively 68%, 95% and 99%.

Problem

Find the mean and standard deviation of the sets of examination (see Table 11.1) marks from the same group of students. The marks are given in order of student name. Histograms were shown in Fig. 11.1. Verify that Chebyshev's theorem holds.

Solution

The mean, E_p, for Programming is given by

$$\begin{aligned} E_p &= (41+42+48+47+49+49+50+52+52 \\ &\quad +14+34+35+37+55+60+67+29+63)/18 \\ &= 45.8. \end{aligned}$$

Table 11.1 Examination marks

Programming																	
41	42	48	47	49	49	50	52	52	14	34	35	37	55	60	67	29	63

Engineering Design																	
15	21	22	30	33	37	39	43	43	47	50	56	60	63	67	70	76	79

Table 11.2 Examination marks, Chebyshev analysis

Range	No. of marks in the range	No. of marks as a percentage
$E_p \pm 2\sigma_p$ (20.8, 70.8)	17	94.4% (> 75%)
$E_e \pm 2\sigma_e$ (9.8, 84.7)	18	100% (> 75%)

In this example we have an accurate evaluation of E_p since we are using results from the whole group. No estimates are used and so we use n rather than $n-1$ in the denominator of the formula for variance, (11.2).

The standard deviation, σ_p, for Programming is given by

$$\begin{aligned}\sigma_p &= \surd\big(\big[(41-45.8)^2 + (42-45.8)^2 + \cdots + (63-45.8)^2\big]/18\big)\\ &= 12.5.\end{aligned}$$

The corresponding figures for Engineering design are $E_e = 47.3$ and $\sigma_e = 9.8$. If we look at the number of marks within ± 2 times the standard deviation of the mean (see Table 11.2) we find that Chebyshev's theorem holds.

Discussion

It is clear from the histograms shown in Fig. 11.1 that for Programming more students are bunched around the mean than for Engineering Design. This is reflected in the very different values of the standard deviations. The greater standard deviation for Engineering Design indicates the greater spread of marks. Although the two sets of marks have similar mean values this does not fully describe the situation.

11.2.5 Covariance and Correlation

If two sets of random variables x and y are being recorded we define their joint variance, or *covariance*, to be $Cov(x, y)$, where

$$Cov(x, y) = E\big[\big(x - E(x)\big)\big(y - E(y)\big)\big].$$

The formula may be justified by considering the values taken by two random variables x and y. If for example, x_1 and y_1 vary in the same direction from the respective means $E(x)$ and $E(y)$, they will be contributing a positive product to the sum of terms

$$\sum_{i=1}^{n}\big(x_i - E(x)\big)\big(y_i - E(x)\big)$$

the larger the magnitude of the product, the stronger the relationship in that particular instance. On the other hand if x_i and y_i vary in opposite directions from their respective means a negative quantity is contributed. If each instance of x, x_i behaved in exactly the same way as the corresponding instance y_i we would have complete agreement and so the expected value of $(x - E(x))(y - E(y))$ would be the positive value of $\sqrt{Var(x)Var(y)}$ on the other hand the complete opposite would produce the negative value. Accordingly we construct a *correlation coefficient* ρ, where

$$\rho = \frac{Cov(x, y)}{\sqrt{Var(x)Var(y)}}$$

as a measure of the joint variability of random variables x and y. The coefficient ρ may take values between ± 1. We speak of variables being *highly correlated*, or on the other hand of being *uncorrelated*, depending on the value of their correlation coefficient ρ.

A value approaching $+1$ corresponds to the case where the joint variability is high. It can be imagined that there is a high correlation between certain life styles and respiratory and other diseases. On the other hand there would seem to be little correlation between the rainfall in Lancashire and the number of delays on the London Underground and so a correlation coefficient closer to zero might be expected. A correlation coefficient approaching -1 suggests an inverse relationship. A possible example might be the number of umbrellas purchased in a holiday resort in any one week against the volume of sun tan lotion. As one goes up, the other is expected to go down. However, it should be emphasised that a coefficient correlation nearing ± 1 does not necessarily imply cause and effect. The whole theory is based on instances of comparable data. Consideration must be given to the quantity and quality of data, the order in which data is compared and to the possibility of coincidence.

Problem

In addition to the previous examination marks consider the marks attained in Fluid Mechanics by the same class. As before the marks are given in order of student name (Table 11.3). Calculate the correlation coefficients for all possible pairings of marks for Programming, Engineering Design and Fluid Mechanics.

Solution

The mean and standard deviation for Fluid Mechanics are found to be 55.6 and 18.6 respectively.

The covariance, $Cov(f, p)$, of the marks for Fluid Mechanics and Programming is found from

Table 11.3 Further examination marks

Fluid Mechanics																	
11	22	38	39	42	51	54	57	60	66	67	68	73	78	76	80	55	63

Table 11.4 Further examination marks, correlation coefficients

	Programming	Engineering Design
Engineering Design	0.18	–
Fluid Mechanics	0.16	0.83

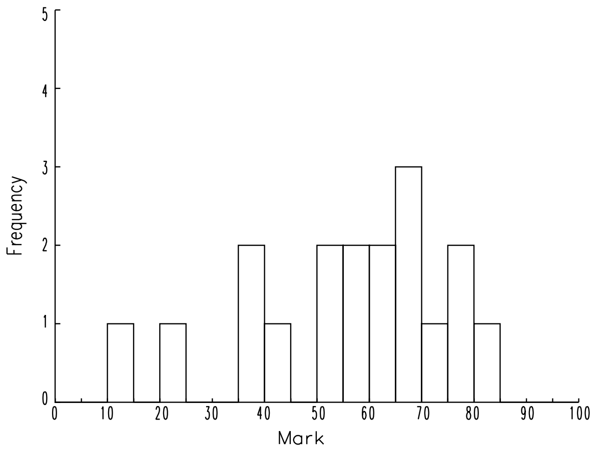


Fig. 11.3 Fluid Mechanics marks

$$\begin{aligned}Cov(f,p) &= \big[(11-55.6)(41-45.8)+(22-55.6)(42-45.8)+\cdots \\ &\quad +(63-55.6)(63-45.8)\big]/18 \\ &= 36.2.\end{aligned}$$

Thus the product $(11-55.6)(41-45.8)$ is the contribution to the sum from the first student, $(22-55.6)(42-45.8)$ from the second, and so on. It is important that the ordering is preserved in order to compare the variability of the examination marks across the class, taking into account how each student performs individually. The correlation coefficient, ρ, of the marks for Fluid Mechanics and Programming is now given by

$$\rho = \frac{38.37}{12.5 \times 18.6} \qquad (11.3)$$

$$= 0.16. \qquad (11.4)$$

Similar calculations for the other pairings produce correlation coefficients are shown in Table 11.4.

Discussion

The table of correlation coefficients shows a relatively high degree of correlation between the marks for Engineering Design and Fluid Mechanics, which is borne out by the histograms in Figs. 11.1 and 11.3. There is a similarity in the profile of the two sets of marks. However this similarity does not necessarily imply that a particular student is likely to have performed similarly in each examination. The similarity may not extend beyond the nature of the examination itself. On the other hand the correlation coefficients between the marks for Programming and the other examinations are low, indicating that as a whole they have little in common. Again this is borne out by consideration of the histograms. Other things being equal perhaps the Programming examination was quite different in character from the other two. The door to speculation is open.

11.3 Least Squares Analysis

We return to the problem of fitting a straight line to a given set of data using the method of least squares first discussed in Chap. 4.

Given data (x_i, y_i), $i = 1, \dots, n$ the aim is to find coefficients a and b such that the straight line $\hat{y} = a + bx$, often referred to as the *regression line*, approximates the data in a least squares sense. In general the data which we present to a least squares program will be subject to error.

In order to measure the reliability of the approximating straight line we use a measure known as the *standard error estimate*, *SEE* defined by

$$SEE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n-2}}$$

where $\hat{y}_i$ are the approximations to the data values y_i calculated using the regression line $\hat{y}_i = a + bx_i$, $i = 1, \dots, n$.

Standard deviation was used to measure dispersion about a mean value. The standard error estimate is a measure of dispersion (or *scatter*) around the regression line. As with standard deviation we take account of the number of degrees of freedom when using standard error estimates. In this case the expression is constrained by the equations for a and b, which are estimates for the actual (and unknown) linear relationship between the variables x and y. The number of degrees of freedom is reduced by two and so $n - 2$ appears in the denominator.

Problem

Fit a regression line (i.e. a least squares linear approximation) to the data given in Table 11.5 using the results from (Chap. 4, question 1) find the standard error estimate. Identify the data points which lie within 1, 2 and 3 standard errors of the regression line.

Table 11.5 Young's Modulus, data

t	0	100	200	300	400	500	600
y	200	190	185	178	170	160	150

Table 11.6 Young's Modulus data, regression line approximations

t	y	$\hat{y}$
0	200	200.2500
100	190	192.2143
200	185	184.1786
300	178	176.1429
400	170	168.1071
500	160	160.0714
600	150	152.0357

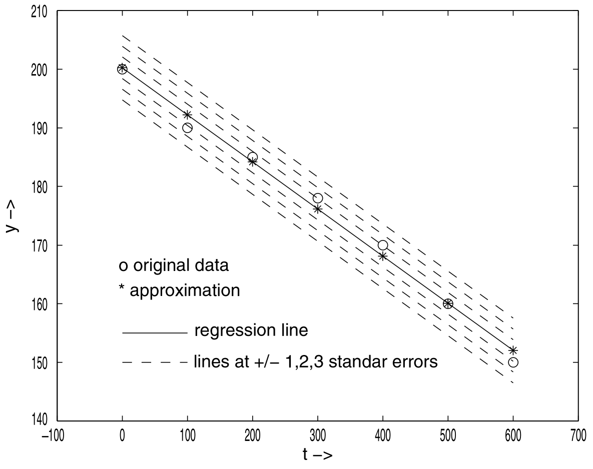


Fig. 11.4 Regression line

Solution

From previous results we know that the regression line is given by $y = 200.25 - 0.0804t$. Approximations $\hat{y}$ to data values y at respective t values are shown in Table 11.6. It follows that the standard error of the estimate *SEE* is given by

$$SEE = \sqrt{\frac{1}{5}(200.2500 - 200)^2 + \cdots + (152.0357 - 150)^2}$$
$$= 1.8342$$

The positions of the data points values (x_i, y_i) are shown in Fig. 11.4. The results show that the data points are distributed around the regression line. As can be seen from Table 11.7 all the points are within ± 2 standard error estimates of the regression line. Since $3/7$ (43%) rather than 68% are within $\pm$ one SEE the distribution is not quite normal, which may suggest that providing all data is collected under similar conditions further data readings are required for a more accurate forecast. Intervals within which we are able to quantify error estimates are known as confidence intervals.

Table 11.7 Youngs's Modulus data, Chebyshev analysis

y	$\hat{y}$	$\lvert \hat{y} - y \rvert \le$	
		± 1.8342	$\pm 2 * 1.8342$
200	200.2500	✓	✓
190	192.2143		✓
185	184.1786	✓	✓
178	176.1429		✓
170	168.1071		✓
160	160.0714	✓	✓
150	152.0357		✓

Discussion

If it is the case that the underlying relationship to which a set of data approximates is linear a further calculation may be made. If we are able to assume that data is collected in such a way that errors are distributed normally with constant variance it is possible to show that σ_b the standard deviation of the coefficient b, in the regression line, $\hat{y} = a + bx$ is given by

$$\sigma_b = \frac{SEE}{\sqrt{S_{xx} - n(E(S_x))^2}}$$

where S_{xx} is the sum of squares of the x_i values and $E(S_x)$ is the mean value of the sum of the n Sx_i values. If applied to the example above the formula would produce $\sigma_b = 0.0035$, which normally would give some indication of the range within which the gradient of the line representing the accurate linear relationship lies.

11.4 Random Numbers

We consider how we might generate a sequence of random numbers, which is useful for a number of reasons.

- Simulation As part of the modelling process computers are often used to simulate situations in which events may happen and which may happen to a varying degree. Flight simulators, road traffic models, economic forecasts, models of industrial and scientific processes and video games are all examples in which a computer program is required to produce events in a random manner in order to simulate the chance events and varying conditions, which happen in everyday life.
- Testing Testing is one of the vital stages in the production of software. Although it is usual to design systematic test data with the intention of tracking every possible route through a program it is all too easy to overlook situations which the user rather than the software designer might encounter. One way of testing for the unpredictable is to produce test data containing a random element.

- Numerical methods There are problems which may be solved by Monte Carlo methods which build a solution based on a number of random evaluations. We look at one such method for numerical integration.

11.4.1 Generating Random Numbers

Short of attaching a coin tossing device to the computer the only feasible way of generating random numbers is by means of software based on mathematical formulae. Since numbers which are related by a formula can no longer be regarded as being truly random, the term *pseudo-random number* is introduced, although the *pseudo* part is often omitted. The skill in constructing such formulae lies in producing sequences which give every appearance of being random. Many have been tried but eventually found to be wanting in that the sequence degenerates into the same relatively short repetitive pattern. Ultimately any formula will return to its starting point and so the aim is to make that interval (the *cycle*) sufficiently large so that for all practical purposes it may be considered to be infinite. Such schemes exclude repetitions within the cycle, which would be a feature of a genuinely random sequence, but this shortcoming is deemed to be acceptable.

The most popular form of random number generator is the *multiplicative congruential* generator which produces integers using the formula

$$n_{i+1} = \lambda n_i \bmod m, \quad i = 1, 2, \ldots \tag{11.5}$$

λ is known as the *multiplier*, m is known as the *modulus* and the starting value n_1 is known as the *seed*. Successive values are found by calculating the remainder when λ times the current value of n_i is divided by m as shown in the following:

Problem

Generate a sequence of random numbers using the multiplicative congruential generator with multiplier 3, modulus 11 and seed 1.

Solution

Using (11.5) we have

$$\begin{aligned}
n_1 &= 1 \\
n_2 &= remainder\big[(3 \times 1) \div 11\big] = 3 \\
n_3 &= remainder\big[(3 \times 3) \div 11\big] = 9 \\
n_4 &= remainder\big[(3 \times 9) \div 11\big] = 5 \\
n_5 &= remainder\big[(3 \times 5) \div 11\big] = 4 \\
n_6 &= remainder\big[(3 \times 4) \div 11\big] = 1
\end{aligned}$$

at which point the cycle 3, 9, 5, 4, 1 repeats.

Discussion

Clearly the sequence we are generating could not be considered to be a satisfactory random selection of integers from the range 1 to 10 since 2, 6, 7, 8 and 10 are missing. On the other hand the sequence $n_{i+1} = 3n_i \bmod 19$ generates all the integers in the range 1 to 18.

If a large selection of numbers is required, repetition of relatively short cycles destroys all pretence that the sequence is chosen at random. The multiplicative formula (11.5) has the potential to generate numbers between 1 and $m - 1$ and so the hunt is on to find a and m to give sequences which generate most, if not all numbers in the range. Ideally m should be as large as possible so that we have an algorithm to cover all eventualities.

11.5 Random Number Generators

We give two popular generators:

$$n_{i+1} = 7^5 n_i \bmod \left(2^{31} - 1\right), \quad i = 1, 2, \ldots \tag{11.6}$$

and

$$n_{i+1} = 13^{13} n_i \bmod 2^{59}, \quad i = 1, 2, \ldots . \tag{11.7}$$

Generator (11.6) has the advantage that it may be programmed on most computers using a high-level language provided care is taken to avoid integer overflow. It has a cycle length of $2^{31} - 1$ (2147483647) and was used in early implementations of Matlab. It has since been replaced by a different algorithm[1] based on multiple seeds with an even longer cycle length of 2^{1430}. Generator (11.7) is not so amenable to high-level programming since the multiplier and modulus are out of normal integer range. However it does have a cycle length of 2^{57}.

11.5.1 Customising Random Numbers

Floating point random numbers may be produced from an integer random number generator having modulus m, by dividing by m. Using the problem above as an example, the output as floating point numbers would be 0.2727, 0.8182, 0.4545, 0.3636, 0.0909 at which point the cycle repeats. Dividing by the modulus, m ensures that numbers produced lie within the range (0, 1), but results may be scaled to lie within any desired range. See the exercises for further details including the production of random integers.

[1] For details see http://www.mathworks.com/moler/random.pdf.

Without further intervention the random numbers we have generated can be considered to be *uniformly* distributed. In other words every number in every possible sequence has an equal chance of appearing. In order to generate a random sequence which has a *normal* distribution having mean, μ and standard deviation, σ random points in the plane would still be generated uniformly but those that do not fall underneath the bell shape (Fig. 11.2) would be rejected.

11.6 Monte Carlo Integration

The Monte Carlo method numerical integration is similar to the method we used for generating normally distributed random numbers. By way of an example consider the following problem, which we know should evaluate to $\frac{\pi}{4}$ (0.7854 to four decimal places), the area of the upper right quadrant of the circle of unit radius centred on the origin.

Problem

Evaluate

$$\int_0^1 \int_0^1 \sqrt{x^2 + y^2}\, dx\, dy.$$

Solution

In the simplest implementation of the method we generate a large number of points (x, y) in the surrounding square of unit area and estimate the integral by counting the proportion that lies within the boundary arc $\sqrt{x^2 + y^2}$ and the axes.

Discussion

The results showing convergence for increasing values of n are shown in Table 11.8. Although easy to apply, the Monte Carlo method can be slow to converge. The method is generally reserved for multiple integrals over a non-standard interval or surface which may be difficult to subdivide. The method may also be used if the

Table 11.8 Monte Carlo estimates for $\pi/4$ (0.7854), question 4

n	Estimate
10^5	0.7868
10^6	0.7860
10^7	0.7852
10^8	0.7854
10^9	0.7854

integrand or its derivatives have discontinuities that invalidate some of the assumptions inherent in the more formal methods of Chap. 5.

Summary This chapter has been a somewhat cursory introduction to statistics. You will have become familiar with a number of technical terms and will be able to appreciate their significance when they are used in other contexts.

We have dealt with the problem of using a computer to generate random numbers. We explained the need for such a facility and showed by example that a multiplicative congruential generator could be used to good effect. We used the aptly named Monte Carlo method to add to the methods for numerical integration given in Chap. 5.

Exercises

1. Use the following segments of code in a program to explore further plotting facilities of Matlab and to verify the results of Sects. 11.2.4 and 11.2.5 using a selection of the Matlab statistical routines *hist* (histogram), *bar* (bar chart), *mean* (mean), *std* (standard deviation) and *cov* (covariance). By default graphs will appear in the Matlab Fig. 1 window, which should be kept in view. Because of the usually more applicable formula (11.2) that Matlab uses for calculating standard deviations, results may differ slightly form those quoted in the chapter.

```
%
% Create vectors p, e and f to contain the marks
% for Programming, Engineering Design and Fluid Mechanics
%
% basic histogram for the Fluid Mechanics
  hist(e)
% set the number of columns to 5, store the number of marks in each column
% in the vector n
% column locations are recorded in cen.
%
  [n cen] = hist(e, 5);
%
% Plot the histogram as a bar chart
%
  bar(cen, n, 'r')            % set the bar colour to r red, (Sect. 1.13)
  axis([10 100 0 6])          % set axes limits
  set(gca,'YTick', 0 : 1 : 6) % set the tick marks on the Y axis
                              % gca is a Matlab abbreviation for get current axis
  gtext('Fluid Mechanics');   % add the title interactively by moving
                              % the cross-hairs and clicking
  xlabel('Mark');             % label the x-axis
  ylabel('Frequency');        % label the y-axis
%
```

```
% Calculate mean values and standard deviations, for example:
%
  mean(p)
  std(p)
%
% Find the covariance of for example Programming v. Engineering Design.
% cov returns a symmetric 2 × 2 matrix with diagonal elements equal to
% the standard deviations of p and e respectively. The covariance
% is given by the off diagonal element.
   X = cov(p, e)
% The correlation coefficient is given by
    X(1,2)/(sqrt(X(1,1)*X(2,2)))
```

2. (i) Write a program to verify the results quoted in Sect. 11.3 relating to fitting the regression line.

   (ii) Repeat the program using data for the *gas* problem as quoted in Table 4.1, Chap. 4. Verify that the standard error estimate, *SEE* is 0.2003 and that the standard deviation of the regression coefficient, σ_b is 0.0958.

   Although the *SEE* value found in case (i) (1.8342) is larger than for case (ii) it should be remembered that in (i) we were dealing with data values of an order 10^2 higher. Scaling the result from (i) by 10^{-2} shows the *SEE* to be smaller by a factor of 10, which is to be expected since the relationship between gas pressure and volume was seen to be essentially non-linear (Fig. 4.1, Chap. 4).

3. (i) Write a program to generate 20 random numbers in the range (0, 1) using formula 11.6. Choose your own seed in the range $(2, 2^{31} - 1)$ but use a large number (for example 123456). You will see by experimenting that a small seed will produce what might be considered to be an unsatisfactory random selection.

   (ii) Modify the program so that the 20 random numbers in the range $(-1, 1)$ are produced. Use the formula $2x - 1$ on numbers x randomly produced in the range $(0, 1)$.

   (iii) Modify the program so that the 10 random integers in the range (5, 100) are produced. Use the formula $95x + 5$ on numbers x in the range (0, 1) to produce values in the range (5, 100) and then take the highest integer number below the random value using the Matlab *floor* function.

   Matlab provides a variety of routines for generating random numbers. In particular the functions *random* and *randint*, which may be used to answer parts (i)–(iii) of the question can take many forms as shown by the following statements:

```
% generate a single number in the range (a, b) from a uniform distribution
  r = random('unif', a, b) ;
% as above but numbers generated in an m by n matrix from the range (a, b)
  A = random('unif', a, b, m, n);
% generate an m by n matrix of integers in the range (a, b)
  A = randint(m, n, [a b]);
```

By default seeds are set automatically in all Matlab functions generating random numbers and will vary from one execution of a program to another. However there are facilities for programmers to set specific seeds.

4. Write a program to verify the results quoted in the chapter for Monte Carlo integration. Estimate the area by generating a given number, n of random points (x, y) drawn from a uniform distribution of x and y values in the range (0, 1.0). All (x, y) points will lie within a rectangle of area equal to 1. As points are generated count the numbers of pairs which are less than or equal to $\sqrt{x^2 + y^2}$, n_1 and those which are greater, n_2. The area of the quadrant will be approximately $n_1/(n_1 + n_2)$.
5. The equation of the bell-shape enclosing points (x, y) over the range $(-\infty < x < \infty)$ having normal distribution having zero mean and standard deviation, σ has the form

$$y = \mathrm{A}\exp\left(\frac{-x^2}{2\sigma^2}\right)$$

where A is a constant. In practice a smaller range may be considered as y values fall rapidly and to all intents and purposes become zero as x values move sufficiently far away from the origin.

Consider the special case, $\mathrm{A} = 1$ and $\sigma = 0.5$. Find the area under the curve for values of x in the three ranges $|x_i| \le i\,\sigma$, $i = 1, 2, 3$. Use the same technique as in the previous question to estimate the areas by generating a given number, N of random points (x, y) drawn from a uniform distribution of x values in the range $(-3, 3)$ and y values in the range $(0, 1)$. All (x, y) points will lie within a rectangle of area equal to 6. There is little point in sampling outside this range of x values since $y(\pm 3)$ is of order 1.5e-8 and decreasing rapidly As points are generated count the number in each of the three groups

$$y \le \exp(-2x^2) \quad \text{and} \quad \text{(i)} \quad |x| \le 0.5, \quad \text{(ii)} \quad |x| \le 1.0, \quad \text{(iii)} \quad |x| \le 1.5.$$

If the final totals are n_1, n_2 and n_3, the corresponding areas will be approximately $3n_i/N$, $i = 1, 2, 3$. Table 11.9 shows the (very slow) progress of the iteration for increasing n. Write a program to verify the results shown in Table 11.9.

Express areas (i), (ii) and (iii) as fractions of the whole area enclosed by the bell-shape and the axis to show values consistent with a normal distribution.

Table 11.9 Monte Carlo estimates for the bell-shape, question 5

N	Area (i)	Area (ii)	Area (iii)
10^4	1.24	1.19	0.84
10^5	1.23	1.18	0.85
10^6	1.25	1.20	0.85
10^7	1.25	1.20	0.86
10^8	1.25	1.20	0.86

Given the decreasing nature of y the integral, the relevant area under the curve namely,

$$\int_{-3}^{+3} \exp\bigl(-2x^2\bigr)\,dx.$$

may be approximated using the standard result

$$\int_{-\infty}^{+\infty} \exp\bigl(-2x^2\bigr)\,dx = \sqrt{\frac{\pi}{2}}.$$

Matlab Index

Table 1 summarises the Matlab supplied operators and functions to which we have referred. In most cases only a few of the options available to the individual functions have been fully utilised. Full details may be found by using the on-line help system. It should be emphasised that this table represents a mere fraction of the facilities available to Matlab users.

Table 1 Principle Matlab operators and functions

Name	Description	Page reference
Arithmetic operators		
+ − ∗ /	Add, subtract, multiply and divide	2
.∗	Element by element matrix multiplication	12
./ .∧	Raising to a power element by element	116
\	Left-division, if $A*X = B$, $X = A^{-1}B$	4
'	Matrix transpose	3
Numerical functions		
:	Generate numbers over a range	6
sqrt, exp, sin, cos etc.	Standard functions	2
bintprog	Binary integer programming	164
eig	Eigenvalues and eigenvectors	229
floor	Nearest integer below a given number	115
function	Define a new function	10
fzero	Root of a function of a single variable	66
linprog	Linear programming	161
lu	LU decomposition	42
max	Largest element	11
fminsearch	Unconstrained local minimum	191
norm	Norm	4

(*continued on the next page*)

Table 1 (Continued)

Name	Description	Page reference
ode45	Solutions to non-stiff differential equations	213
optimset	Sets options for optimisation routines	68
pi	π	3
polyfit	Fit polynomial to data	92
polyval	Polynomial evaluation	93
qr	QR factorisation	229
quad	Numerical integration	114
roots	Roots of a polynomial	3
size	Number of rows and columns	8
fsolve	Solve systems of nonlinear equations	68
strcmp	Compare two strings	10
strncmp	Compare first n characters of two strings	10
trapz	Numerical integration using the trapezium rule	115
Program control		
@< *name* >	Passing function names as parameters	66
&& || > <	Relationships: and, or less than, greater than,	7
<= >= ~ ==	less than or equal, greater than or equal, not, equal	7
break	Terminate a loop prematurely	68
if	Conditional command execution	7
clear	Delete all or specified program variables	16
for loop	Program section to be repeated n times	6
load	Retrieve program variables from backing store	15
save	Save program variables to backing store	15
while loop	Program section to be repeated conditionally	8
Miscellaneous		
%	Program comment, not regarded by Matlab as a command	6
	also used in specifying formats	13
...	Continuation mark	13
Ctrl C	Terminate execution	8
more	Page by page or continuous screen output	7
Data types		
cell	Extension of matrix concept to elements of different type	14
complex	Specify a complex number	132
global	Declare a variable to be global	132
i	Complex number i	132
real	Extract the real part of a complex number	133
single	Use 32-bit rather than 64-bit word	42
syms	Symbolic variables	133

(*continued on the next page*)

Table 1 (Continued)

Name	Description	Page reference
disp	Display text string or program variable	9
ezplot	Function plotter	133
fopen	Access a file	14
format	Define a format	12
fprintf	format a string for output to a file	14
frewind	File rewind	15
input	Request keyboard input	9
sprintf	Formatted screen output	12
textscan	formatted file input	14
type	File listing	14
Plotting		
axis	Set axis limits	160
colorbar	Colour chart	12
figure	Start a new (plotting) figure	161
gca	Get access to the current plot to change settings	245
gtext	Interactive graph annotating facility	161
linspace	Specify points over an interval	12
meshgrid	Specify points over a grid	12
plot	2D-graph plotting	11
surf	3D-surface plotting	12
Special matrices		
eye	Generate an identity matrix	165
diag	Diagonal matrix with or without sub-diagonals	214
hilb	Hilbert matrix	43
ones	Generate a matrix with all elements = 1	43
zeros	Generate a matrix with all elements = 0	214
Statistical functions		
cov	Covariance	245
mean	Mean	11
std	Standard deviation	11
bar	Bar chart	11
pie	Pie chart	11
hist	Histogram	245
polyfit	Linear regression	92
rand	Random number generator	246
randint	Random integer generator	246

Index