4 Differentiation and Integration

Differentiation and integration are the fundamental operations of differential calculus and occur in almost every field of mathematics, science, and engineering. Determining the derivative of a function analytically may be tedious but is relatively straightforward. The inverse of this process, that of determining the integral of a function, can often be difficult analytically or even impossible.

The difficulty of determining the analytical integral for certain functions has encouraged the development of many numerical procedures for determining approximately the value of definite integrals. In many situations the procedures work well because integration is a smoothing process and errors in the approximation tend to cancel each other. However, for certain types of functions, difficulties may arise and these will be examined as part of our discussion of specific numerical methods for the approximate evaluation of definite integrals.

4.1 Introduction

In the next section of this chapter we show how the derivative of a function may be estimated for a particular value of the independent variable. The numerical approximations for derivatives require only function values. These approximations can be used to great advantage when derivatives are required in a program. Their application saves the program user the task of determining the analytical expressions for these derivatives. In Section 4.3 and beyond we introduce the reader to a range of numerical integration methods, including methods suitable for infinite ranges of integration. Generally numerical integration works well, but there are pathological integrals that will defeat the best numerical algorithms.

4.2 Numerical Differentiation

In this section we present a range of approximations for first- and higher-order derivatives. Before we derive these approximations in detail we give a simple example that illustrates the dangers of the careless or naive use of such derivative approximations. The simplest approximation for the first-order derivative of a given function $f(x)$ arises from the formal definition of the derivative:

$$\frac{df}{dx} = \lim_{h\to 0}\left(\frac{f(x+h)-f(x)}{h}\right) \tag{4.1}$$

One interpretation of (4.1) is that the derivative of a function $f(x)$ is the slope of the tangent to the function at the point x. For small h we obtain the approximation to the derivative:

$$\frac{df}{dx} \approx \left(\frac{f(x+h) - f(x)}{h}\right) \tag{4.2}$$

This would appear to imply that the smaller the value of h, the better the value of our approximation in (4.2). The following MATLAB script plots Figure 4.1, which shows the error for various values of h.

```
% e3s401.m
g = @(x) x.^9;
x = 1; h(1) = 0.5;
hvals = [ ]; dfbydx = [ ];
for i = 1:17
    h = h/10;
    b = g(x); a = g(x+h);
    hvals = [hvals h];
    dfbydx(i) = (a-b)/h;
end;
exact = 9;
loglog(hvals,abs(dfbydx-exact),'*')
axis([1e-18 1 1e-8 1e4])
xlabel('h value'), ylabel('Error in approximation')
```

Figure 4.1 shows that for large values of h the error is large but falls rapidly as h is decreased. However, when h becomes less than about 10^{-9}, rounding errors dominate and the approximation becomes much worse. Clearly care must be taken in the choice of h. With this warning in mind we develop methods of differing accuracies for any order derivative.

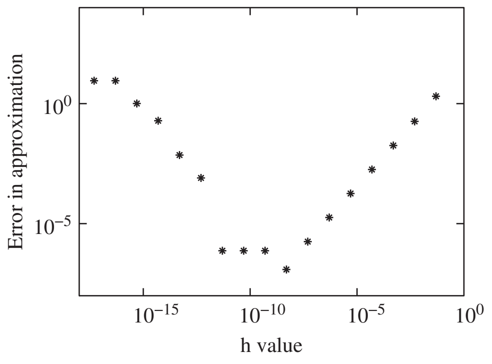

FIGURE 4.1 A log-log plot showing the error in a simple derivative approximation.

We have seen how a simple approximate formula for the first derivative can be easily obtained from the formal definition of the derivative. However, it is difficult to approximate higher derivatives and deduce more accurate formulae in this way; instead we will use the Taylor series expansion of the function $y = f(x)$. To determine the central difference approximation for the derivative of this function at x_i we expand $f(x_i + h)$:

$$f(x_i + h) = f(x_i) + hf'(x_i) + (h^2/2!)f''(x_i) + (h^3/3!)f'''(x_i) + (h^4/4!)f^{(iv)}(x_i) + \cdots \tag{4.3}$$

We sample $f(x)$ at points a distance h apart and write $x_i + h$ as x_{i+1}, and so on. We will also write $f(x_i)$ as f_i and $f(x_{i+1})$ as f_{i+1}. Thus

$$f_{i+1} = f_i + hf'(x_i) + (h^2/2!)f''(x_i) + (h^3/3!)f'''(x_i) + (h^4/4!)f^{(iv)}(x_i) + \cdots \tag{4.4}$$

Similarly

$$f_{i-1} = f_i - hf'(x_i) + (h^2/2!)f''(x_i) - (h^3/3!)f'''(x_i) + (h^4/4!)f^{(iv)}(x_i) - \cdots \tag{4.5}$$

We can find an approximation to the first derivative as follows. Subtracting (4.5) from (4.4) gives

$$f_{i+1} - f_{i-1} = 2hf'(x_i) + 2\left(h^3/3!\right)f'''(x_i) + \cdots$$

Thus, neglecting terms in h^3 and higher, we have

$$f'(x_i) = \left(f_{i+1} - f_{i-1}\right)/2h \quad \text{with errors of} \quad O\left(h^2\right) \tag{4.6}$$

This is the central difference approximation and differs from (4.2), which is a forward difference approximation. Equation (4.6) is more accurate than (4.2) but in the limit as h approaches zero, the two are identical.

To determine an approximation for the second derivative we add (4.4) and (4.5) to obtain

$$f_{i+1} + f_{i-1} = 2f_i + 2\left(h^2/2!\right)f''(x_i) + 2\left(h^4/4!\right)f^{(iv)}(x_i) + \cdots$$

Thus, neglecting terms in h^4 and higher, we have

$$f''(x_i) = \left(f_{i+1} - 2f_i + f_{i-1}\right)/h^2 \quad \text{with errors of} \quad O\left(h^2\right) \tag{4.7}$$

By taking more terms in the Taylor series, together with the Taylor series for $f(x + 2h)$ and $f(x - 2h)$, and so on, and performing similar manipulations, we can obtain higher derivatives and more accurate approximations if required. Table 4.1 gives examples of these formulae.

Table 4.1 Derivative Approximations

	Multipliers for $f_{i-3} \cdots f_{i+3}$							
	f_{i-3}	f_{i-2}	f_{i-1}	f_i	f_{i+1}	f_{i+2}	f_{i+3}	**Order of Error**
$2hf'(x_i)$	0	0	−1	0	1	0	0	h^2
$h^2f''(x_i)$	0	0	1	−2	1	0	0	h^2
$2h^3f'''(x_i)$	0	−1	2	0	−2	1	0	h^2
$h^4f^{(iv)}(x_i)$	0	1	−4	6	−4	1	0	h^2
$12hf'(x_i)$	0	1	−8	0	8	−1	0	h^4
$12h^2f''(x_i)$	0	−1	16	−30	16	−1	0	h^4
$8h^3f'''(x_i)$	1	−8	13	0	−13	8	−1	h^4
$6h^4f^{(iv)}(x_i)$	−1	12	−39	56	−39	12	−1	h^4

The MATLAB function `diffgen` defined in the following computes the first, second, third, and fourth derivative of a given function with errors of $O(h^4)$ for a specified value of x using data from the table.

```
function q = diffgen(func,n,x,h)
% Numerical differentiation.
% Example call: q = diffgen(func,n,x,h)
% Provides nth order derivatives, where n = 1 or 2 or 3 or 4
% of the user defined function func at the value x, using a step h.
if (n==1)|(n==2)|(n==3)|(n==4)
    c = zeros(4,7);
    c(1,:) = [ 0 1 -8 0 8 -1 0];
    c(2,:) = [ 0 -1 16 -30 16 -1 0];
    c(3,:) = [1.5 -12 19.5 0 -19.5 12 -1.5];
    c(4,:) = [ -2 24 -78 112 -78 24 -2];
    y = feval(func,x+[-3:3]*h);
    q = c(n,:)*y.';  q = q/(12*h^n);
else
    disp('n must be 1, 2, 3 or 4'), return
end
```

For example,

```
result = diffgen('cos',2,1.2,0.01)
```

determines the second derivative of $\cos(x)$ for $x = 1.2$ with $h = 0.01$ and gives `-0.3624` for the result. The following script calls the function `diffgen` four times to determine the first four derivatives of $y = x^7$ when $x = 1$:

```
% e3s402.m
g = @(x) x.^7;
h = 0.5; i = 1;
```

```
disp('     h       1st deriv  2nd deriv   3rd deriv    4th deriv');
while h>=1e-5
    t1 = h;
    t2 = diffgen(g, 1, 1, h);
    t3 = diffgen(g, 2, 1, h);
    t4 = diffgen(g, 3, 1, h);
    t5 = diffgen(g, 4, 1, h);
    fprintf('%10.5f %10.5f %10.5f %11.5f %12.5f\n',t1,t2,t3,t4,t5);
    h = h/10; i = i+1;
end
```

The output from the preceding script is

```
   h       1st deriv  2nd deriv   3rd deriv    4th deriv
0.50000    1.43750   38.50000   191.62500    840.00000
0.05000    6.99947   41.99965   209.99816    840.00000
0.00500    7.00000   42.00000   210.00000    840.00001
0.00050    7.00000   42.00000   210.00000    839.97579
0.00005    7.00000   42.00000   209.98521   -290.13828
```

Note that as h is decreased the estimates for the first and second derivatives steadily improve, but when $h = 5 \times 10^{-4}$ the estimate for the fourth derivative begins to deteriorate. When $h = 5 \times 10^{-5}$ the estimate for the third derivative also begins to deteriorate and the fourth derivative is very inaccurate. In general we cannot predict when this deterioration will begin. It should be noted that different platforms may give different results for this value.

4.3 Numerical Integration

We will begin by examining the definite integral

$$I = \int_a^b f(x)\,dx \tag{4.8}$$

The evaluation of such integrals is often called quadrature and we will develop methods for both finite and infinite values of a and b.

The definite integral (4.8) is a summation process but it may also be interpreted as the area under the curve $y = f(x)$ from a to b. Any areas above the x-axis are counted as positive; any areas below the x-axis are counted as negative. Many numerical methods for integration are based on using this interpretation to derive approximations to the integral. Typically the interval $[a, b]$ is divided into a number of smaller subintervals, and by making simple approximations to the curve $y = f(x)$ in the subinterval, the area of the subinterval may be obtained. The areas of all the subintervals are then summed to give an approximation to the integral in the interval $[a, b]$. Variations of this technique are developed by taking groups of subintervals and fitting different degree polynomials to approximate $y = f(x)$ in each of these groups. The simplest of these methods is the trapezoidal rule.

The trapezoidal rule is based on the idea of approximating the function $y = f(x)$ in each subinterval by a straight line so that the shape of the area in the subinterval is trapezoidal. Clearly, as the number of subintervals used increases, the straight lines will approximate the function more closely. Dividing the interval from a to b into n subintervals of width h (where $h = (b-a)/n$) we can calculate the area of each subinterval since the area of a trapezium is its base times the mean of its heights. These heights are f_i and f_{i+1} where $f_i = f(x_i)$. Thus the area of the trapezium is

$$h\left(f_i + f_{i+1}\right)/2 \quad \text{for} \quad i = 0, 1, 2, \ldots, n-1$$

Summing all the trapezia gives the composite trapezoidal rule for approximating (4.8):

$$I \approx h\left\{\left(f_0 + f_n\right)/2 + f_1 + f_2 + \cdots + f_{n-1}\right\} \tag{4.9}$$

The truncation error, which is the error due to the implicit approximation in the trapezoidal rule, is

$$E_n \leq \left(b-a\right)h^2 M/12 \tag{4.10}$$

where M is the upper bound for $|f''(t)|$ and t must be in the range a to b. The MATLAB function `trapz` implements this procedure and we use it in Section 4.4 to compare the performance of the trapezoidal rule with the more accurate Simpson's rule.

The level of accuracy obtained from a numerical integration procedure is dependent on three factors. The first two are the nature of the approximating function and the number of intervals used. These are controlled by the user and give rise to the truncation error, that is, the error inherent in the approximation. The third factor influencing accuracy is the rounding error, the error caused by the fact that practical computation has limited precision. For a particular approximating function the truncation error will decrease as the number of subintervals increases. Integration is a smoothing process and rounding errors do not present a major problem. However, when many intervals are used, the time to solve the problem becomes more significant because of the increased amount of computation. This problem may be reduced by writing the script efficiently.

4.4 Simpson's Rule

Simpson's rule is based on using a quadratic polynomial approximation to the function $f(x)$ over a pair of subintervals; it is illustrated in Figure 4.2. If we integrate the quadratic polynomial passing through the points (x_0, f_0), (x_1, f_1), (x_2, f_2), where $f_1 = f(x_1)$, and so on,

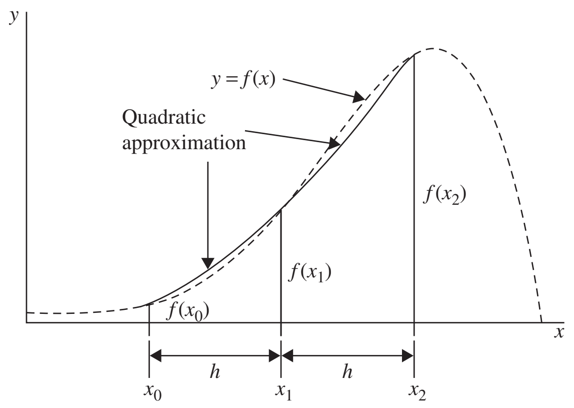

FIGURE 4.2 Simpson's rule, using a quadratic approximation over two intervals.

the following formula is obtained:

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{h}{3}\left(f_0 + 4f_1 + f_2\right) \tag{4.11}$$

This is Simpson's rule for one pair of intervals. Applying the rule to all pairs of intervals in the range a to b and adding the results produces the following expression, known as the composite Simpson's rule:

$$\int_{a}^{b} f(x)\,dx = \frac{h}{3}\left\{f_0 + 4\left(f_1 + f_3 + f_5 + \cdots + f_{2n-1}\right) + 2\left(f_2 + f_4 + \cdots + f_{2n-2}\right) + f_{2n}\right\} \tag{4.12}$$

Here n indicates the number of pairs of intervals and $h = (b-a)/(2n)$. The composite rule may be also be written as a vector product:

$$\int_{a}^{b} f(x)\,dx = \frac{h}{3}\left(\mathbf{c}^{\top}\mathbf{f}\right) \tag{4.13}$$

where $\mathbf{c} = [1\ 4\ \ 2\ \ 4\ \ 2\ldots 2\ \ 4\ \ 1]^{\top}$ and $\mathbf{f} = [f_1\ f_2\ f_3\ \ldots\ f_{2n}]^{\top}$.

The error arising from the approximation, called the truncation error, is approximated by

$$E_n = (b-a)h^4 f^{(iv)}(t)/180$$

where t lies between a and b. An upper bound for the error is given by

$$E_n \le (b-a)h^4 M/180 \tag{4.14}$$

where M is an upper bound for $|f^{(iv)}(t)|$. The upper bound for the error in the simpler trapezoidal rule, (4.10), is proportional to h^2 rather than h^4. This makes Simpson's rule superior to the trapezoidal rule in terms of accuracy at the expense of more function evaluations.

To illustrate different ways of implementing Simpson's rule we provide two alternatives, `simp1` and `simp2`. The function `simp1` creates a vector of coefficients `v` and a vector of function values `y` and multiplies the two vectors together. Function `simp2` provides a more conventional implementation of Simpson's rule. In each case the user must provide the definition of the function to be integrated, the lower and upper limits of integration, and the number of subintervals to be used. The number of subintervals must be an even number since the rule fits a function to a *pair* of subintervals.

```
function q = simp1(func,a,b,m)
% Implements Simpson's rule using vectors.
% Example call: q = simp1(func,a,b,m)
% Integrates user defined function func from a to b, using m divisions
if (m/2)~=floor(m/2)
    disp('m must be even'); return
end
h = (b-a)/m; x = a:h:b;
y = feval(func,x);
v = 2*ones(m+1,1);  v2 = 2*ones(m/2,1);
v(2:2:m) = v(2:2:m)+v2;
v(1) = 1;  v(m+1) = 1;
q = (h/3)*y*v;
```

The second nonvectorized form of this function is

```
function q = simp2(func,a,b,m)
% Implements Simpson's rule using for loop.
% Example call: q = simp2(func,a,b,m)
% Integrates user defined function
% func from a to b, using m divisions
if (m/2) ~= floor(m/2)
    disp('m must be even'); return
end
h = (b-a)/m;
s = 0; yl = feval(func,a);
for j = 2:2:m
    x = a+(j-1)*h;  ym = feval(func,x);
    x = a+j*h;   yh = feval(func,x);
    s = s+yl+4*ym+yh;  yl = yh;
end
q = s*h/3;
```

The following script calls either `simp1` or `simp2`. These functions can be used to demonstrate the effect on accuracy of the number of pairs of intervals used. The script evaluates the integral of x^7 in the range 0 to 1.

```
% e3s403.m
n = 4; i = 1;
tic
disp('    n integral value')
while n < 1025
    simpval = simp1(@(x) x.^7,0,1,n); % or simpval = simp2(etc.);
    fprintf('%5.0f %15.12f\n',n,simpval)
    n = 2*n; i = i+1;
end
t = toc;
fprintf('\ntime taken = %6.4f secs\n',t)
```

The output from this script using `simp1` is as follows:

```
  n integral value
     4  0.129150390625
     8  0.125278472900
    16  0.125017702579
    32  0.125001111068
    64  0.125000069514
   128  0.125000004346
   256  0.125000000272
   512  0.125000000017
  1024  0.125000000001

time taken = 0.0635 secs
```

On running this script, but using `simp2`, we obtain the same values for the integral but the following results for the time taken:

```
time taken = 0.1335 secs
```

Equation (4.14) shows that the truncation error will decrease rapidly for values of h smaller than 1. The preceding results illustrate this. The rounding error in Simpson's rule is due to evaluating the function $f(x)$ and the subsequent multiplications and additions. Note also that the vectorized version, `simp1`, is a little faster than `simp2`.

We now evaluate the same integral using the MATLAB function `trapz`. To call this function the user must provide a vector of function values `f`. The function `trapz(f)` estimates the integral of the function assuming unit spacing between the data points. Thus to determine the integral we multiply `trapz(f)` by the increment `h`.

```
% e3s404.m
n = 4; i = 1; f = @(x) x.^7;
tic
disp('    n  integral value')
while n<1025
    h = 1/n; x = 0:h:1;
    trapval = h*trapz(f(x));
    fprintf('%5.0f %15.12f\n',n,trapval)
    n = 2*n; i = i+1;
end
t = toc;
fprintf('\ntime taken = %4.2f secs\n',t)
```

Running this script gives

```
    n  integral value
    4  0.160339355469
    8  0.134043693542
   16  0.127274200320
   32  0.125569383381
   64  0.125142397981
  128  0.125035602755
  256  0.125008900892
  512  0.125002225236
 1024  0.125000556310

time taken = 0.06 secs
```

These results illustrate the fact that the trapezoidal rule is less accurate than the Simpson rule.

4.5 Newton–Cotes Formulae

Simpson's rule is an example of a Newton–Cotes formula for integration. Other examples of these formulae can be obtained by fitting higher-degree polynomials through the appropriate number of points. In general we fit a polynomial of degree n through $n+1$ points. The resulting polynomial can then be integrated to provide an integration formula. Here are some examples of Newton–Cotes formulae together with estimates of their truncation errors.

For $n=3$ we have

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}\left(f_0+3f_1+3f_2+f_3\right) + \text{truncation error } \frac{3h^5}{80}f^{iv}(t) \tag{4.15}$$

where t lies in the interval x_0 to x_3.

For $n = 4$ we have

$$\int_{x_0}^{x_4} f(x)\,dx = \frac{2h}{45}\left(7f_0 + 32f_1 + 12f_2 + 32f_3 + 7f_4\right) + \text{truncation error } \frac{8h^7}{945} f^{(vi)}(t) \tag{4.16}$$

where t lies in the interval x_0 to x_4. Composite rules can be generated for both rules (4.15) and (4.16). The truncation errors indicate that some improvement in accuracy may be obtained by using these rules rather than Simpson's rule. However, the rules are more complex; consequently, greater computational effort is involved and rounding errors may become a more significant problem.

The MATLAB function `quad` uses an adaptive recursive Simpson's rule and the function `quadl` uses adaptive Lobatto quadrature. The MATLAB function `quadgk` uses an adaptive Gauss–Kronrod rule, which is particularly efficient for smooth and oscillatory integrals. The limits of integration may be infinite.

The performance of `quad` and `quadl` and `simp1` (called with both 1024 and 4096 panels) are compared by determining the error when evaluating the integral e^x from 0 to n where $n = 2.5 : 2.5 : 25$ using the following script:

```
% e3s405.m
for n = 1:10; n1 = 2.5*n;
    ext = exp(n1)-1;
    err(n,1) = simp1('exp',0,n1,1024)-ext;
    err(n,2) = simp1('exp',0,n1,4096)-ext;
    err(n,3) = quadl('exp',0,n1)-ext;
    err(n,4) = quad('exp',0,n1)-ext;
end
err
```

Running this script gives the following:

```
err =
  2.2062e-012  8.8818e-015  7.2414e-009  3.9510e-009
  4.6552e-010  1.8190e-012  1.0203e-011  8.6445e-009
  2.8889e-008  1.1323e-010  2.9315e-009  1.4057e-008
  1.1129e-006  4.3437e-009  4.5475e-010  1.6258e-008
  3.3101e-005  1.2928e-007            0  1.5891e-008
  8.3618e-004  3.2666e-006 -9.3132e-010  1.8626e-008
  1.8872e-002  7.3716e-005            0  7.4506e-009
  3.9221e-001  1.5321e-003 -5.9605e-008            0
  7.6535e+000  2.9899e-002  9.5367e-007  9.5367e-007
  1.4211e+002  5.5516e-001 -1.5259e-005 -1.5259e-005
```

These results show the advantage of using adaptive subinterval sizes. Simpson's rule has a fixed interval size. For the smaller ranges of integration it performs very well but as the range of integration increases, accuracy decreases. Generally the adaptive methods maintain a much higher level of accuracy.

4.6 Romberg Integration

A major problem that arises with the nonadaptive Simpson's or Newton–Cotes rule is that the number of intervals required to provide the required accuracy is initially unknown. Clearly one approach to this problem is to double successively the number of intervals used and compare the results of applying a particular rule, as illustrated by the examples in Section 4.4. Romberg's method provides an organized approach to this problem and utilizes the results obtained by applying Simpson's rule with different interval sizes to reduce the truncation error.

Romberg integration may be formulated as follows. Let I be the exact value of the integral and T_i the approximate value of the integral obtained using Simpson's rule with i intervals. Consequently, we may write an approximation for the integral I that includes contributions from the truncation error as follows (note that the error terms are expressed in powers of h^4):

$$I = T_i + c_1 h^4 + c_2 h^8 + c_3 h^{12} + \cdots \tag{4.17}$$

If we double the number of intervals, h is halved, giving

$$I = T_{2i} + c_1 \left(h/2\right)^4 + c_2 \left(h/2\right)^8 + c_3 \left(h/2\right)^{12} + \cdots \tag{4.18}$$

We can eliminate the terms in h^4 by subtracting (4.17) from 16 times (4.18), giving

$$I = \left(16T_{2i} - T_i\right)/15 + k_2 h^8 + k_3 h^{12} + \cdots \tag{4.19}$$

Notice that the dominant or most significant term in the truncation error is now of order h^8. In general this will provide a significantly improved approximation to I. For the remainder of this discussion it is advantageous to use a double subscript notation. If we generate an initial set of approximations by successively halving the interval we may represent them by $T_{0,k}$ where $k = 0, 1, 2, 3, 4, \ldots$. These results may be combined in a similar manner to that described in (4.19) by using the general formula

$$T_{r,k} = \left(16^r T_{r-1,k+1} - T_{r-1,k}\right) / \left(16^r - 1\right) \quad \text{for} \quad k = 0,\ 1,\ 2,\ 3\ \ldots \quad \text{and} \quad r = 1,\ 2,\ 3,\ \ldots \tag{4.20}$$

Here r represents the current set of approximations we are generating. The calculations may be tabulated as follows:

$$\begin{array}{lllll}
T_{0,0} & T_{0,1} & T_{0,2} & T_{0,3} & T_{0,4} \\
T_{1,0} & T_{1,1} & T_{1,2} & T_{1,3} & \\
T_{2,0} & T_{2,1} & T_{2,2} & & \\
T_{3,0} & T_{3,1} & & & \\
T_{4,0} & & & &
\end{array}$$

In this case, the interval has been halved four times to generate the first five values in the table denoted by $T_{0,k}$. The preceding formula for $T_{r,k}$ is used to calculate the remaining values in the table and at each stage the order of the truncation error is increased by four. A common alternative is to write the preceding table with the rows and columns interchanged.

At each stage the interval size is given by

$$(b-a)/2^k \quad \text{for} \quad k = 0, 1, 2, \ldots \tag{4.21}$$

Romberg integration is implemented in the following MATLAB function, `romb`:

```
function [W T] = romb(func,a,b,d)
% Implements Romberg integration.
% Example call: W = romb(func,a,b,d)
% Integrates user defined function func from a to b, using d stages.
T = zeros(d+1,d+1);
for k = 1:d+1
    n = 2^k;  T(1,k) = simp1(func,a,b,n);
end
for p = 1:d
    q = 16^p;
    for k = 0:d-p
        T(p+1,k+1) = (q*T(p,k+2)-T(p,k+1))/(q-1);
    end
end
W = T(d+1,1);
```

We now apply the function `romb` to the evaluation of $x^{0.1}$ in the range 0 to 1. The call of the function `romb` is

```
>> [integral table] = romb(@(x) x.^0.1,0,1,5)
```

Calling this function gives the following output. Note that the best estimate is the single value in the last row of the table.

```
integral =
    0.9066


table =
    0.7887    0.8529    0.8829    0.8969    0.9034    0.9064
    0.8572    0.8849    0.8978    0.9038    0.9066         0
    0.8850    0.8978    0.9038    0.9066         0         0
    0.8978    0.9038    0.9066         0         0         0
    0.9038    0.9066         0         0         0         0
    0.9066         0         0         0         0         0
```

This integral is a surprisingly difficult one and obtaining an accurate result presents a significant problem. The exact solution to four decimal places is 0.9090 so the application of the Romberg method gives only two places of accuracy. However, taking $n = 10$ does give the answer correct to four places:

```
>> integral = romb(@(x) x.^0.1,0,1,10)

integral =
    0.9090
```

Generally the Romberg method is very efficient and accurate. For example, it evaluates the integral of e^x from 0 to 10 using five divisions of the interval more accurately and slightly more quickly than the function `quad` with the default tolerance.

An interesting exercise for the reader is to convert the function `romb` to work with the MATLAB function `trapz` instead of `simp1`.

4.7 Gaussian Integration

The common feature of the methods considered so far is that the integrand is evaluated at equal intervals within the range of integration. In contrast, Gaussian integration requires the evaluation of the integrand at specified, but unequal, intervals. For this reason Gaussian integration cannot be applied to data values that are sampled at equal intervals of the independent variable. The general form of the rule is

$$\int_{-1}^{1} f(x)\,dx = \sum_{i=1}^{n} A_i f(x_i) \tag{4.22}$$

The parameters A_i and x_i are chosen so that, for a given n, the rule is exact for polynomials up to and including degree $2n - 1$. It should be noticed that the range of integration is required to be from -1 to 1. This does not restrict the integrals to which Gaussian integration can be applied since if $f(x)$ is to be integrated in the range a to b, then it can be replaced by the function $g(t)$ integrated from -1 to 1 where

$$t = (2x - a - b) \,/\, (b - a)$$

Note that in the preceding formula, when $x = a$, $t = -1$ and when $x = b$, $t = 1$.

We will now determine the four parameters A_i and x_i for $n = 2$ in (4.22). Thus (4.22) now becomes

$$\int_{-1}^{1} f(x)\,dx = A_1 f(x_1) + A_2 f(x_2) \tag{4.23}$$

This integration rule will be exact for polynomials up to and including degree 3 by ensuring that the rule is exact for the polynomials 1, x, x^2, and x^3 in turn. Thus four equations are obtained as follows:

$$\begin{aligned}
f(x) = 1 &\quad \text{gives} \quad \int_{-1}^{1} 1dx = 2 = A_1 + A_2 \\
f(x) = x &\quad \text{gives} \quad \int_{-1}^{1} xdx = 0 = A_1x_1 + A_2x_2 \\
f(x) = x^2 &\quad \text{gives} \quad \int_{-1}^{1} x^2dx = 2/3 = A_1x_1^2 + A_2x_2^2 \\
f(x) = x^3 &\quad \text{gives} \quad \int_{-1}^{1} x^3dx = 0 = A_1x_1^3 + A_2x_2^3
\end{aligned} \tag{4.24}$$

Solving these equations gives

$$x_1 = -1/\surd 3, \quad x_2 = 1/\surd 3, \quad A_1 = 1, \quad A_2 = 1$$

Thus

$$\int_{-1}^{1} f(x)\, dx = f\left(-\frac{1}{\surd 3}\right) + f\left(\frac{1}{\surd 3}\right) \tag{4.25}$$

Notice that this rule, like Simpson's rule, is exact for cubic equations but requires fewer function evaluations.

A general procedure for obtaining the values of A_i and x_i is based on the fact that in the range of integration it can be shown that $x_1, x_2, \ldots, x_n$ are the roots of the Legendre polynomial of degree n. The values of A_i can then be obtained from an expression involving the Legendre polynomial of degree n, evaluated at x_i. Tables have been produced for the values of x_i and A_i for various values of n; see Abramowitz and Stegun (1965) and Olver et al. (2010). Abramowitz and Stegun provide an excellent reference not only for these functions but for a very extensive range of mathematical functions. However, this classic work is now becoming outdated and a newer handbook of mathematical functions for the twenty-first century by Olver et al. has been published with many improvements, for example, clearer, color graphics. However, this new text contains far fewer tables of functions, since most can now be rapidly computed on a personal computer.

The function `fgauss` defined as follows performs Gaussian integration. It includes a substitution so that integration in the range a to b is converted to an integration in the range -1 to 1.

```
function q = fgauss(func,a,b,n)
% Implements Gaussian integration.
% Example call: q = fgauss(func,a,b,n)
% Integrates user defined function func from a to b, using n divisions
% n must be 2 or 4 or 8 or 16.
if (n==2)|(n==4)|(n==8)|(n==16)
    c = zeros(8,4);  t = zeros(8,4);
    c(1,1) = 1;
    c(1:2,2) = [.6521451548; .3478548451];
    c(1:4,3) = [.3626837833; .3137066458; .2223810344; .1012285362];
    c(:,4 )= [.1894506104; .1826034150; .1691565193; .1495959888; ...
            .1246289712; .0951585116; .0622535239; .0271524594];
    t(1,1) = .5773502691;
    t(1:2,2) = [.3399810435; .8611363115];
    t(1:4,3) = [.1834346424; .5255324099; .7966664774; .9602898564];
    t(:,4) = [.0950125098; .2816035507; .4580167776; .6178762444; ...
            .7554044084; .8656312023; .9445750230; .9894009350];
    j = 1;
    while j<=4
        if 2^j==n; break;
        else
            j = j+1;
        end
    end
    s = 0;
    for k = 1:n/2
        x1 = (t(k,j)*(b-a)+a+b)/2;
        x2 = (-t(k,j)*(b-a)+a+b)/2;
        y = feval(func,x1)+feval(func,x2);
        s = s+c(k,j)*y;
    end
    q = (b-a)*s/2;
else
    disp('n must be equal to 2, 4, 8 or 16'); return
end
```

The following script calls the function `fgauss` to integrate $x^{0.1}$ from 0 to 1.

```
% e3s406.m
disp('  n  integral value');
for j = 1:4
    n = 2^j;
    int = fgauss(@(x) x.^0.1,0,1,n);
    fprintf('%3.0f %14.9f\n',n,int)
end
```

The output of this script is

```
 n  integral value
 2    0.916290737
 4    0.911012914
 8    0.909561226
16    0.909199952
```

Gaussian integration with $n = 16$ gives a better result than that obtained by Romberg's method with five divisions of the interval.

4.8 Infinite Ranges of Integration

Other formulae of the Gauss type are available to allow us to deal with integrals having a special form and infinite ranges of integration. These are the Gauss–Laguerre and Gauss–Hermite formulae and they take the following forms.

4.8.1 Gauss–Laguerre Formula

This method is developed from the following equation:

$$\int_0^\infty e^{-x} g(x)dx = \sum_{i=1}^{n} A_i g(x_i) \tag{4.26}$$

The parameters A_i and x_i are chosen so that, for a given n, the rule is exact for polynomials up to and including degree $2n-1$. Considering the case when $n = 2$, we have

$$\begin{aligned}
g(x) = 1 \quad &\text{gives} \quad \int_0^\infty e^{-x}dx = 1 = A_1 + A_2\\
g(x) = x \quad &\text{gives} \quad \int_0^\infty xe^{-x}dx = 1 = A_1x_1 + A_2x_2\\
g(x) = x^2 \quad &\text{gives} \quad \int_0^\infty x^2e^{-x}dx = 2 = A_1x_1^2 + A_2x_2^2\\
g(x) = x^3 \quad &\text{gives} \quad \int_0^\infty x^3e^{-x}dx = 6 = A_1x_1^3 + A_2x_2^3
\end{aligned} \tag{4.27}$$

Having evaluated the integrals on the left side of equations (4.27) we may solve for the four unknowns x_1, x_2, A_1, and A_2 so that (4.26) becomes

$$\int_0^\infty e^{-x} g(x) dx = \frac{2+\sqrt{2}}{4} g(2-\sqrt{2}) + \frac{2-\sqrt{2}}{4} g(2+\sqrt{2})$$

It can be shown that the x_i are the roots of the nth-order Laguerre polynomial and the coefficients A_i can be calculated from an expression involving the derivative of an nth-order Laguerre polynomial evaluated at x_i.

In general we wish to evaluate integrals of the form

$$\int_0^\infty f(x)\, dx$$

We may write this integral as

$$\int_0^\infty e^{-x}\{e^x f(x)\}\, dx$$

Thus, using (4.26), we have

$$\int_0^\infty f(x)\ dx = \sum_{i=1}^{n} A_i \exp(x_i) f(x_i) \tag{4.28}$$

Equation (4.28) allows integrals to be evaluated over an infinite range, assuming that the value of the integral is finite.

The Gauss–Laguerre method is implemented by the MATLAB function `galag`:

```
function s = galag(func,n)
% Implements Gauss-Laguerre integration.
% Example call: s = galag(func,n)
% Integrates user defined function func from 0 to inf
% using n divisions. n must be 2 or 4 or 8.
if (n==2)|(n==4)|(n==8)
    c = zeros(8,3);  t = zeros(8,3);
    c(1:2,1) = [1.533326033; 4.450957335];
    c(1:4,2) = [.8327391238; 2.048102438; 3.631146305; 6.487145084];
    c(:,3) = [.4377234105; 1.033869347; 1.669709765; 2.376924702;...
              3.208540913; 4.268575510; 5.818083368; 8.906226215];
    t(1:2,1) = [.5857864376; 3.414213562];
    t(1:4,2) = [.3225476896; 1.745761101; 4.536620297; 9.395070912];
    t(:,3) = [.1702796323; .9037017768; 2.251086630; 4.266700170;...
              7.045905402; 10.75851601; 15.74067864; 22.86313174];
```

```
        j = 1;
        while j<=3
            if 2^j==n; break
            else
                j = j+1;
            end
        end
        s = 0;
        for k = 1:n
            x = t(k,j); y = feval(func,x);
            s = s+c(k,j)*y;
        end
    else
        disp('n must be 2, 4 or 8'); return
    end
```

Sample values x_i and the product $A_i \exp(x_i)$ are given in the function definition. A more complete list may be found in Abramowitz and Stegun (1965) and Olver et al. (2010).

We will now evaluate the integral $\log_e(1+e^{-x})$ from zero to infinity. The following script evaluates the integral using the function `galag`.

```
% e3s407.m
disp(' n   integral value');
for j = 1:3
    n = 2^j;
    int = galag(@(x) log(1+exp(-x)),n);
    fprintf('%3.0f%14.9f\n',n,int)
end
```

The output is as follows:

```
n   integral value
 2   0.822658694
 4   0.822358093
 8   0.822467051
```

Note that the exact result is $\pi^2/12 = 0.82246703342411$. The eight-point integration formula is accurate to six decimal places!

4.8.2 Gauss–Hermite Formula

This method is developed from the following equation:

$$\int_{-\infty}^{\infty} \exp(-x^2)g(x)dx = \sum_{i=1}^{n} A_i g(x_i) \tag{4.29}$$

Again, the parameters A_i and x_i are chosen so that, for a given n, the rule is exact for polynomials up to and including degree $2n-1$. For the case $n=2$ we have

$$\begin{aligned}
g(x) &= 1 \quad \text{gives} \quad \int_{-\infty}^{\infty} \exp(-x^2)dx = \sqrt{\pi} = A_1 + A_2 \\
g(x) &= x \quad \text{gives} \quad \int_{-\infty}^{\infty} x\exp(-x^2)dx = 0 = A_1x_1 + A_2x_2 \\
g(x) &= x^2 \quad \text{gives} \quad \int_{-\infty}^{\infty} x^2\exp(-x^2)dx = \frac{\sqrt{\pi}}{2} = A_1x_1^2 + A_2x_2^2 \\
g(x) &= x^3 \quad \text{gives} \quad \int_{-\infty}^{\infty} x^3\exp(-x^2)dx = 0 = A_1x_1^3 + A_2x_2^3
\end{aligned} \tag{4.30}$$

We have evaluated the integrals on the left side of equations (4.30) and may now solve for the four unknowns x_1, x_2, A_1, and A_2 so that (4.29) becomes

$$\int_{-\infty}^{\infty} \exp(-x^2)g(x)dx = \frac{\sqrt{\pi}}{2}g\left(-\frac{1}{\sqrt{2}}\right) + \frac{\sqrt{\pi}}{2}g\left(\frac{1}{\sqrt{2}}\right)$$

An alternative approach is to note that x_i are the roots of the nth-order Hermite polynomial $H_n(x)$. The coefficients A_i can then be determined from an expression involving the derivative of the nth-order Hermite polynomial evaluated at x_i.

In general we wish to evaluate integrals of the form

$$\int_{-\infty}^{\infty} f(x)\,dx$$

We may write this integral as

$$\int_{-\infty}^{\infty} \exp\left(-x^2\right)\left\{\exp(x^2)f(x)\right\}dx$$

and using (4.29) we have

$$\int_{-\infty}^{\infty} f(x)dx = \sum_{i=1}^{n} A_i \exp\left(x_i^2\right)f(x_i) \tag{4.31}$$

Again, care must be taken to apply (4.31) only to functions that have a finite integral in the range $-\infty$ to ∞. Extensive tables of x_i and A_i are given in Abramowitz and Stegun (1965) and Olver et al. (2010). The MATLAB function gaherm implements Gauss–Hermite integration:

```
function s = gaherm(func,n)
% Implements Gauss-Hermite integration.
% Example call: s = gaherm(func,n)
% Integrates user defined function func from -inf to +inf,
% using n divisions. n must be 2 or 4 or 8 or 16
if (n==2)|(n==4)|(n==8)|(n==16)
    c = zeros(8,4);  t = zeros(8,4);
    c(1,1) = 1.461141183;
    c(1:2,2) = [1.059964483; 1.240225818];
    c(1:4,3) = [.7645441286; .7928900483; .8667526065; 1.071930144];
    c(:,4) = [.5473752050; .5524419573; .5632178291; .5812472754; ...
              .6097369583; .6557556729; .7382456223; .9368744929];
    t(1,1) = .7071067811;
    t(1:2,2) = [.5246476233; 1.650680124];
    t(1:4,3) = [.3811869902; 1.157193712; 1.981656757; 2.930637420];
    t(:,4) = [.2734810461; .8229514491; 1.380258539; 1.951787991; ...
              2.546202158; 3.176999162; 3.869447905; 4.688738939];
    j = 1;
    while j<=4
        if 2^j==n; break;
        else
            j = j+1;
        end
    end
    s=0;
    for k = 1:n/2
        x1 = t(k,j); x2 = -x1;
        y = feval(func,x1)+feval(func,x2);
        s = s+c(k,j)*y;
    end
else
    disp('n must be equal to 2, 4, 8 or 16'); return
end
```

We will now evaluate the integral

$$\int_{-\infty}^{\infty} \frac{dx}{(1+x^2)^2}$$

by the Gauss–Hermite method. The following script uses `gaherm` to integrate this function.

```
% e3s408.m
disp(' n   integral value');
for j = 1:4
    n = 2^j;
    int = gaherm(@(x) 1./(1+x.^2).^2,n);
    fprintf('%3.0f%14.9f\n',n,int)
end
```

The results from running this script are

```
n   integral value
 2   1.298792163
 4   1.482336098
 8   1.550273058
16   1.565939612
```

The exact value of this integral is $\pi/2 = 1.570796\ldots$

4.9 Gauss–Chebyshev Formula

We now consider two interesting cases where the sample points x_i and weights w_i are known in a closed or analytical form. The two integrals together with their closed forms are

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}}\,dx = \frac{\pi}{n}\sum_{k=1}^{n} f(x_k) \quad \text{where} \quad x_k = \cos\left(\frac{(2k-1)\pi}{2n}\right) \tag{4.32}$$

$$\int_{-1}^{1} \sqrt{1-x^2} f(x)dx = \frac{\pi}{n+1}\sum_{k=1}^{n} \sin^2\left(\frac{k\pi}{n+1}\right) f(x_k) \quad \text{where} \quad x_k = \cos\left(\frac{k\pi}{n+1}\right) \tag{4.33}$$

These expressions are members of the Gauss family, in this case variations of the Gauss–Chebyshev formula. Clearly it is extremely easy to use these formulae for integrands of the required form that have a specified $f(x)$. It is simply a matter of evaluating the function at the specified points, multiplying by the appropriate factor, and summing these products. A MATLAB script or function can easily be developed and is left as an exercise for the reader (see Problem 4.11).

4.10 Gauss–Lobatto Integration

Lobatto integration or quadrature (Abramowitz and Stegun, 1965) is named after Dutch mathematician Rehuel Lobatto. It is similar to Gaussian quadrature, which we discussed previously, but the integration points include the end points of the integration interval. This has an advantage when the procedure is used in a subinterval because data can be shared between consecutive subintervals. However, Lobatto quadrature is less accurate than the Gaussian formula.

Lobatto quadrature of function $f(x)$ on interval $[-1\ 1]$ is given by the formula

$$\int_{-1}^{1} f(x)dx = \frac{2}{n(n-1)}\left[f(1)+f(-1)\right] + \sum_{i=2}^{n-1} w_i f(x_i) + R_n$$

Here the points x_i are the roots of the Legendre polynomial $P_{n-1}(x) = 0$. The weights other than for $f(1)$ and $f(-1)$, which both equal $2/(n(n-1))$, are calculated from the following formula:

$$w_i = \frac{2}{n(n-1)[P_{n-1}(x_i)]^2} \quad (x_i \neq \pm 1)$$

Clearly from this description it is an easy matter to calculate the weights required if the roots of the derivative of the Legendre polynomial are found.

The coefficients of any order Legendre polynomial can be found using Bonnet's recursion formula

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

where $P_0(x) = 1$, $P_1(x) = x$, and $P_n(x)$ is the nth Legendre polynomial. Alternatively, a recurrence relation for the polynomials can be found using the differential equation definition of the Legendre function.

The following MATLAB function is based on generating the polynomial coefficients using a recurrence formula and then finding the roots of the derivative of this polynomial using the MATLAB function `roots`. The range has been converted to any range a to b.

```
function Iv = lobattof(func,a,b,n)
% Implementation of Lobatto's method
% func is the function to be integrated from the a to b
% using n points.
% Generate Legendre polynomials based on recurrence relation
% derived from the differential equation which the Legendre polynomial
% satisfies.
% Obtain derivitive of that polynomial
% The roots of this polynomial give the Lobatto nodes
% From the nodes calculate the weights using standard algorithm
lc = [ ];
for k = 0:n-1
    if n>=2*k
        fnk = factorial(2*n-2*k);
        fnp = 2^n*factorial(k)*factorial(n-k)*factorial(n-2*k);
        lc(n-2*k+1) = (-1)^k*fnk/fnp;
    end
end
% Find coefficients of derivitive of the polynomial
lcd = [ ];
for k = 0:n-1
    if n>=2*k
        lcd(n-2*k+1) = (n-2*k)*lc(n-2*k+1);
    end
end
lcd(n) = 0;
% Obtain Lobatto points
x = roots(fliplr(lcd(2:n+1)));
x1 = sort(x,'descend');
pv = zeros(size(x));
% Calculate Lobatto weights
for k = 1:n+1
    pv = pv+lc(k)*x.^(k-1);
end
n = n+1;
w = 2./(n*(n-1)*pv.^2);
w = [2/(n*(n-1)); w; 2/(n*(n-1))];
% Transform to range a to b
x1 = (x*(b-a)+(a+b))/2;
pts = [a; x1; b];
% Implement rule for integration
Iv = (b-a)*w'*feval(func,pts)/2;
```

To test the function the following MATLAB script is used to integrate $f(x) = e^{5x}\cos(2x)$ from 0 to $\pi/2$.

```
% e3s414.m
g = @(x) exp(5*x).*cos(2*x); a = 0; b = pi/2;
```

```
for n = [2 4 8 16 32 64]
    Iv = lobattof(g,a,b,n);
    fprintf('%3.0f%19.9f\n',n,real(Iv))
end
exact = -5*(exp(2.5*pi)+1)/29;
fprintf('\n Exact %15.9f\n',exact)
```

This gives the following results:

```
  2     -674.125699610
  4     -443.869707406
  8     -444.305258005
 16     -444.305258027
 32     -444.307194507
 64      -16.994770727

Exact  -444.305258034
```

Note that as the number of points used is increased up to 16, the integration becomes more accurate. However, above this value the accuracy decreases. This is because the function `lobattof` determines the abscissae weights by finding the roots of a polynomial. This becomes less accurate as n increases.

An alternative approach to determine the value of an integral is to subdivide the range of integration into subintervals and then apply a Lobatto rule with a small number of points to each subinterval. The following function allows the user to choose the number of points in the Lobatto integration and the number of subintervals in which the Lobatto integration is applied.

```
function s = lobattomp(func,a,b,n,m)
% n is the number of points in the Labatto quadrature
% m is the number of subintervals of the range of the integration.
h = (b-a)/m; s = 0;
for panel = 0:m-1
    a0 =a+panel*h; b0 = a+(panel+1)*h;
    s = s+lobattof(func,a0,b0,n);
end
```

The following script evaluates the error in the integration of $e^{5x}\cos(2x)$ over the range 0 to $\pi/2$. The script considers a 4-, 5-, ..., 8-point Lobatto integration applied to subintervals, the number of subintervals ranging from 2, 4, 8 to 256.

```
% e3s415.m
g = @(x) exp(5*x).*cos(2*x); a = 0; b = pi/2;
format short e
m = 2; k = 0;
while m<512
    % m is number of panels, k is the index
    k = k+1;
    p = 0;
    for n = 4:8
        % n number of Labotto points, p is index
        p = p+1;
        Integral_err(k,p) = real(lobattomp(g,a,b,n,m))+5*(exp(2.5*pi)+1)/29;
    end
    m = 2*m;
end
Integral_err
```

Running this script gives the following output. Each row gives the value of the error for the specified number of subintervals, beginning with 2, 4, 8, and so on to 256, and each column gives the value of the error for the specified number of points in the Lobatto integration, from 4, 5, ..., 8.

```
Integral_err =
  1.5122e-002  2.6320e-004  1.6910e-006  1.8372e-009 -3.7573e-011
  1.0050e-004  3.5484e-007  4.4201e-010  3.4106e-013 -1.7053e-012
  4.4719e-007  3.7181e-010  3.4106e-013  5.6843e-013 -1.0800e-012
  1.8037e-009  1.1369e-013  1.1369e-013  5.1159e-013 -1.0232e-012
  7.0486e-012 -2.2737e-013  2.2737e-013  5.1159e-013 -9.0949e-013
 -5.6843e-014 -2.2737e-013  2.8422e-013  5.1159e-013 -9.0949e-013
 -1.1369e-013 -2.2737e-013  2.2737e-013  6.2528e-013 -9.0949e-013
 -1.1369e-013 -2.8422e-013  2.2737e-013  6.2528e-013 -6.8212e-013
```

It is evident that increasing the number of subintervals (m) and increasing the number of points in the Lobatto integration (n) reduces the error in the integration. However, when the number of points in the Labatto integration and the number of subintervals increase beyond certain values the accuracy of the integration begins to decrease. The values of m and n at which this happens is problem dependent.

A further disadvantage of the Gauss formula is that the location and weight of the abscissae change as their number increases. For example, suppose we have evaluated an integral using an n-point Gauss quadrature rule. To increase the accuracy we could now increase the number of points and use the Gauss rule again, but all the points would be at a new location. An alternative strategy is to keep the existing n points and add to them $n+1$ points located at the best positions. This is the Kronrod method (Kronrod, 1965).

Thus a three-point Gauss method can be extended by keeping the three points and adding four more to give a seven-point rule. The MATLAB function `quadgk` implements adaptive Gauss–Kronrod quadrature.

A discussion of the family of Gaussian quadrature methods is given by Thompson (2010).

4.11 Filon's Sine and Cosine Formulae

These formulae can be applied to integrals of the form

$$\int_a^b f(x)\cos kx\,dx \quad \text{and} \quad \int_a^b f(x)\sin kx\,dx \tag{4.34}$$

The formulae are generally more efficient than standard methods for this form of integral. To derive the Filon formulae we first consider an integral of the form

$$\int_0^{2\pi} f(x)\cos kx\,dx$$

By the method of undetermined coefficients we can obtain an approximation to this integrand as follows. Let

$$\int_0^{2\pi} f(x)\ \cos x\,dx = A_1 f(0) + A_2 f(\pi) + A_3 f(2\pi) \tag{4.35}$$

Requiring that this should be exact for $f(x) = 1$, x, and x^2, we have

$$\begin{aligned} 0 &= A_1 + A_2 + A_3 \\ 0 &= A_2\pi + A_3 2\pi \\ 4\pi &= A_2\pi^2 + A_3 4\pi^2 \end{aligned}$$

Thus $A_1 = 2/\pi$, $A_2 = -4/\pi$, and $A_3 = 2/\pi$. Thus,

$$\int_0^{2\pi} f(x)\ \cos x\,dx = \frac{1}{\pi}[2f(0) - 4f(\pi) + 2f(2\pi)] \tag{4.36}$$

More general results can be developed as follows:

$$\int_0^{2\pi} f(x)\cos kx\,dx = h[A\{f(x_n)\sin kx_n - f(x_0)\sin kx_0\} + BC_e + DC_o]$$

$$\int_0^{2\pi} f(x)\ \sin kx\,dx = h[A\{f(x_0)\cos\,kx_0 - f(x_n)\cos kx_n\} + BS_e + DS_o]$$

where $h = (b-a)/n$, $q = kh$ and

$$A = \left(q^2 + q\sin 2q/2 - 2\sin^2 q\right)/q^3 \tag{4.37}$$

$$B = 2\left\{q\left(1+\cos^2 q\right) - \sin 2q\right\}/q^3 \tag{4.38}$$

$$D = 4\left(\sin q - q\,\cos q\right)/q^3$$

$$C_o = \sum_{i=1,\,3,\,5\ldots}^{n-1} f(x_i)\cos kx_i \tag{4.39}$$

$$C_e = \frac{1}{2}\{f(x_0)\cos kx_0 + f(x_n)\cos kx_n\} + \sum_{i=2,4,6\ldots}^{n-2} f(x_i)\cos kx_i$$

C_o and C_e are odd and even sums of cosine terms. S_o and S_e are similarly defined with respect to sine terms.

It is important to note that Filon's method, when applied to functions of the form given in (4.34), usually gives better results than Simpson's method for the same number of intervals. Approximations may be used for the expressions for A, B, and D given in (4.37), (4.38), and (4.11) by expanding them in series of ascending powers of q. This leads to the following results:

$$A = 2q^2\left(q/45 - q^3/315 + q^5/4725 - \cdots\right)$$

$$B = 2\left(1/3 + q^2/15 - 2q^4/105 + q^6/567 - \cdots\right)$$

$$D = 4/3 - 2q^2/15 + q^4/210 - q^6/11340 + \cdots$$

When the number of intervals becomes very large, h and hence q become small. As q tends to zero, A tends to zero, B tends to 2/3, and D tends to 4/3. Substituting these values into the formula for Filon's method, it can be shown that it becomes equivalent to Simpson's rule. However, in these circumstances the accuracy of Filon's rule may be worse than Simpson's rule owing to the additional complexity of the calculations.

The MATLAB function `filon` implements Filon's method for the evaluation of appropriate integrals. In the parameter list, function `func` defines $f(x)$ of (4.34) and this is multiplied by cos kx when `cas = 1` or sin kx when `cas ~= 1`. The parameters `l` and `u` specify the lower and upper limit of the integral and `n` specifies the number of divisions required. The script incorporates a modification to the standard Filon method such that the series approximation is used if q is less than 0.1 rather than (4.37) to (4.11). The justification for this is that as q becomes small, the accuracy of series approximation is sufficient and easier to compute.

```
function int = filon(func,cas,k,l,u,n)
% Implements filon's integration.
% Example call: int = filon(func,cas,k,l,u,n)
% If cas = 1, integrates cos(kx)*f(x) from l to u using n divisions.
% If cas ~= 1, integrates sin(kx)*f(x) from l to u using n divisions.
% User defined function func defines f(x).
if (n/2)~=floor(n/2)
    disp('n must be even'); return
else
    h = (u-l)/n; q = k*h;
    q2 = q*q; q3 = q*q2;
    if q<0.1
        a = 2*q2*(q/45-q3/315+q2*q3/4725);
        b = 2*(1/3+q2/15+2*q2*q2/105+q3*q3/567);
        d = 4/3-2*q2/15+q2*q2/210-q3*q3/11340;
    else
        a = (q2+q*sin(2*q)/2-2*(sin(q))^2)/q3;
        b = 2*(q*(1+(cos(q))^2)-sin(2*q))/q3;
        d = 4*(sin(q)-q*cos(q))/q3;
    end
    x = l:h:u;
    y = feval(func,x);
    yodd = y(2:2:n);  yeven = y(3:2:n-1);
    if cas == 1
        c = cos(k*x);
        codd = c(2:2:n);  co = codd*yodd';
        ceven = c(3:2:n-1);
        ce = (y(1)*c(1)+y(n+1)*c(n+1))/2;
        ce = ce+ceven*yeven';
        int = h*(a*(y(n+1)*sin(k*u)-y(1)*sin(k*l))+b*ce+d*co);
    else
        s = sin(k*x);
        sodd = s(2:2:n);  so = sodd*yodd';
        seven = s(3:2:n-1);
        se = (y(1)*s(1)+y(n+1)*s(n+1))/2;
        se = se+seven*yeven';
        int = h*(-a*(y(n+1)*cos(k*u)-y(1)*cos(k*l))+b*se+d*so);
    end
end
```

We now test the function `filon` by integrating $\sin x/x$ in the range 1×10^{-10} to 1. The lower limit is set at 1×10^{-10} to avoid the singularity at zero.

The following script uses `filon` and `filonmod` to evaluate the integral. The function `filonmod` removes the ability to switch to the series formula in `filon`. Note that from (4.34), we define $f(x) = 1/x$ for this particular problem.

```
% e3s409.m
n = 4;
g = @(x) 1./x;
disp('  n  Filon no switch  Filon with switch');
while n<=4096
    int1 = filonmod(g,2,1,1e-10,1,n);
    int2 = filon(g,2,1,1e-10,1,n);
    fprintf('%4.0f %17.8e %17.8e\n',n,int1,int2)
    n = 2*n;
end
```

Running this script gives

```
   n  Filon no switch  Filon with switch
   4    1.72067549e+006   1.72067549e+006
   8    1.08265940e+005   1.08265940e+005
  16    6.77884667e+003   6.77884667e+003
  32    4.24742208e+002   4.24742207e+002
  64    2.74361110e+001   2.74361124e+001
 128    2.60175423e+000   2.60175321e+000
 256    1.04956252e+000   1.04956313e+000
 512    9.52549009e-001   9.52550585e-001
1024    9.46489412e-001   9.46487290e-001
2048    9.46109716e-001   9.46108334e-001
4096    9.46085291e-001   9.46084649e-001
```

The exact value of the integral is 0.9460831.

In this particular problem, the switch occurs when $n = 16$. The preceding output shows that the values of the integral obtained with the switch are marginally more accurate. However, it should be noted that experiments carried out by us have shown that for a lower accuracy of computation than that supplied in the MATLAB environment, the accuracy of Filon's method, including the switch, is significantly better. The reader may find it interesting to experiment with the value of q at which the switch occurs. This is currently set at 0.1.

Finally we choose a function that is appropriate for Filon's method and compare the results with Simpson's rule. The function is $\exp(-x/2)\cos(100x)$ integrated between 0 and 2π.

The MATLAB script that implements this comparison is

```
% e3s410.m
n = 4;
disp('   n   Simpsons value   Filons value');
g1 = @(x) exp(-x/2);
g2 = @(x) exp(-x/2).*cos(100*x);
while n<=2048
    int1 = filon(g1,1,100,0,2*pi,n);
    int2 = simp1(g2,0,2*pi,n);
    fprintf('%4.0f %17.8e %17.8e\n',n,int2,int1)
    n = 2*n;
end
```

The results of this comparison are

```
   n   Simpsons value   Filons value
   4   1.91733833e+000   4.55229440e-005
   8  -5.73192992e-001   4.72338540e-005
  16   2.42801799e-002   4.72338540e-005
  32   2.92263624e-002   4.76641931e-005
  64  -8.74419731e-003   4.77734109e-005
 128   5.55127202e-004   4.78308678e-005
 256  -1.30263888e-004   4.78404787e-005
 512   4.53408415e-005   4.78381786e-005
1024   4.77161559e-005   4.78381120e-005
2048   4.78309107e-005   4.78381084e-005
```

The exact value of the integral to 10 significant digits is $4.783810813 \times 10^{-5}$. In this particular problem the switch to the series approximations does not take place because of the high value of the coefficient k. The output shows that using 2048 intervals, Filon's method is accurate to eight significant digits. In contrast, Simpson's rule is accurate to only five significant digits and its behavior is highly erratic. However, timing the evaluation of this integral shows that Simpson's method is about 25% faster than Filon's method.

4.12 Problems in the Evaluation of Integrals

The methods outlined in the previous sections are based on the assumption that the function to be integrated is well behaved. If this is not so, then the numerical methods may give poor, or totally useless, results. Problems may occur if

1. The function is continuous in the range of integration but its derivatives are discontinuous or singular.
2. The function is discontinuous in the range of integration.

3. The function has singularities in the range of integration.
4. The range of integration is infinite.

It is vital that these conditions are identified because in most cases these problems cannot be dealt with directly by numerical techniques. Consequently, some preparation of the integrand is required before the integral can be evaluated by the appropriate numerical method. Case 1 is the least serious condition but since the derivatives of polynomials are continuous, polynomials cannot accurately represent functions with discontinuous derivatives. Ideally, the discontinuity or singularity in the derivative should be located and the integral split into a sum of two or more integrals. The procedure is the same in case 2; the position of the discontinuities must be found and the integral split into a sum of two or more integrals, the ranges of which avoid the discontinuities. Case 3 can be dealt with in various ways: using a change of variable, integration by parts, and splitting the integral. In case 4 we must use a method suitable for an infinite range of integration (see Section 4.8) or make a substitution.

The following integral, taken from Fox and Mayers (1968), is an example of case 4:

$$I = \int_1^\infty \frac{dx}{x^2 + \cos(x^{-1})} \tag{4.40}$$

This integral can be estimated either by using function `galag` (using the substitution $y = x - 1$ to give a lower limit of zero) or by substituting $z = 1/x$. Thus $dz = -dx/x^2$ and (4.40) may be transformed as follows:

$$I = -\int_1^0 \frac{dz}{1 + z^2 \cos(z)} \quad \text{or} \quad I = \int_0^1 \frac{dz}{1 + z^2 \cos(z)} \tag{4.41}$$

The integral (4.41) can easily be evaluated by any standard method.

We have discussed a number of techniques for numerical integration. It must be said, however, that even the best methods have difficulty with functions that change very rapidly for small changes in the independent variable. An example of this type of function is $\sin(1/x)$. A MATLAB plot of this function is shown in Section 3.8. However, this plot does not give a true representation of the function in the range -0.1 to 0.1 because in this range the function is changing very rapidly and the number of plotting points and the screen resolution are inadequate. Indeed, as x tends to zero the frequency of the function tends to infinity. A further difficulty is that the function has a singularity at $x = 0$. If we decrease the range of x, then a small section of the function can be plotted and displayed. For example, in the range $x = 2 \times 10^{-4}$ to 2.05×10^{-4} there are approximately 19 cycles of the function $\sin(1/x)$, as shown in Figure 4.3, and in this limited range the function can be effectively sampled and plotted. Summarizing, the value of this function can change from an extreme positive to an extreme negative value for a relatively small change in x. The consequence

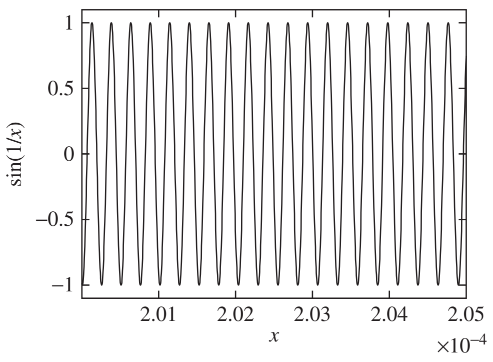

FIGURE 4.3 The function $\sin(1/x)$ in the range $x = 2 \times 10^{-4}$ to 2.05×10^{-4}. Nineteen cycles of the function are displayed.

of this is that when estimating the integral of the function, a great number of divisions of the range of integration are needed to provide the required level of accuracy, particularly for smaller values of x. For this type of problem adaptive integration methods, such as that used by the MATLAB function `quadl`, have been introduced. These methods increase the number of intervals only in those regions where the function is changing very rapidly, thus reducing the overall number of calculations required.

4.13 Test Integrals

We now compare the Gauss and Simpson methods of integration with the MATLAB function `quadl` using the following integrals:

$$\int_0^1 x^{0.001} dx = 1000/1001 = 0.999000999\ldots \tag{4.42}$$

$$\int_0^1 \frac{dx}{1+(230x-30)^2} = (\tan^{-1} 200 + \tan^{-1} 30)/230 = 0.0134924856495 \tag{4.43}$$

$$\int_0^4 x^2(x-1)^2(x-2)^2(x-3)^2(x-4)^2 dx = 10240/693 = 14.776334776 \tag{4.44}$$

To generate the comparative results we define the function `ftable` as follows:

```
function y = ftable(fname,lowerb,upperb)
% Generates table of results.
```

```
intg = fgauss(fname,lowerb,upperb,16);
ints = simp1(fname,lowerb,upperb,2048);
intq = quadl(fname,lowerb,upperb,.00005);
fprintf('%19.8e %18.8e %18.8e \n',intg,ints,intq)
```

The following script applies this function to the three test integrals:

```
% e3s411.m
clear
disp('function      Gauss               Simpson              quadl')
fprintf('Func 1'), ftable(@(x) x.^0.001,0,1)
fprintf('Func 2'), ftable(@(x) 1./(1+(230*x-30).^2),0,1)
g = @(x) (x.^2).*((1-x).^2).*((2-x).^2).*((3-x).^2).*((4-x).^2);
fprintf('Func 3'), ftable(g,0,4)
```

The output from this script is

```
function      Gauss               Simpson              quadl
Func 1    9.99003302e-001   9.98839883e-001   9.98981017e-001
Func 2    1.46785776e-002   1.34924856e-002   1.34925421e-002
Func 3    1.47763348e+001   1.47763348e+001   1.47763348e+001
```

The integrals (4.42) and (4.43) are difficult to evaluate and Figure 4.4 shows plots of the integrands in the range of integration. Each function, at some point, changes rapidly with small changes of the independent variable, making such functions extremely difficult to integrate numerically if a high degree of accuracy is required.

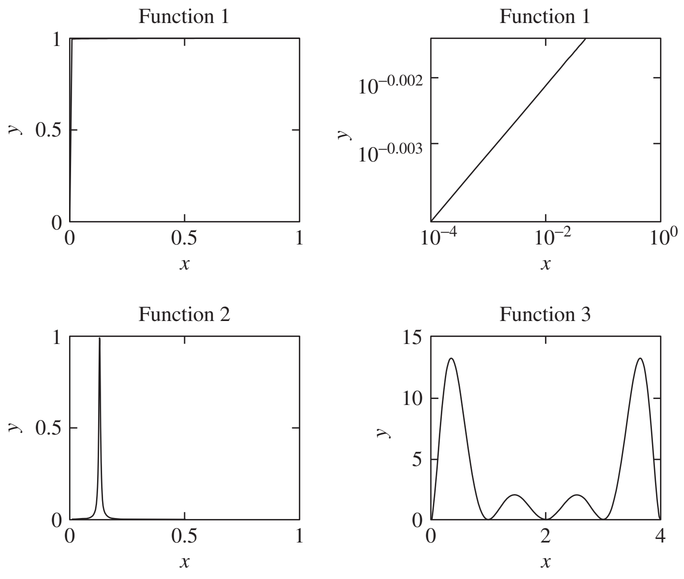

FIGURE 4.4 Plots of functions defined in script `e3s411`.

4.14 Repeated Integrals

In this section we confine ourselves to a discussion of repeated integrals using two variables. It is important to note that there is a significant difference between double integrals and repeated integrals. However, it can be shown that if the integrand satisfies certain requirements then double integrals and repeated integrals are equal in value. A detailed discussion of this result is given in Jeffrey (1979).

We have considered in this chapter various techniques for evaluating single integrals. The extension of these methods to repeated integrals can present considerable scripting difficulties. Furthermore, the number of computations required for the accurate evaluation of a repeated integral can be enormous. While many algorithms for the evaluation of single integrals can be extended to repeated integrals, here only extensions to the Simpson and Gauss methods with two variables are presented. These have been chosen as the best compromise between programming simplicity and efficiency.

An example of a repeated integral is

$$\int_{a_1}^{b_1} dx \int_{a_2}^{b_2} f(x,y)\,dy \tag{4.45}$$

In this notation the function is integrated with respect to x from a_1 to b_1 and with respect to y from a_2 to b_2. Here the limits of integration are constant but in some applications they may be variables.

4.14.1 Simpson's Rule for Repeated Integrals

We now apply Simpson's rule to the repeated integral (4.45) by applying it first in the y direction and then in the x direction. Consider three equispaced values of y: y_0, y_1, and y_2. On applying Simpson's rule, (4.11), to integration with respect to y in (4.45), we have

$$\int_{x_0}^{x_2} dx \int_{y_0}^{y_2} f(x,y)dy \approx \int_{x_0}^{x_2} k\left\{f(x,y_0) + 4f(x,y_1) + f(x,y_2)\right\}/3\,dx \tag{4.46}$$

where $k = y_2 - y_1 = y_1 - y_0$.

Consider now three equispaced values of x: x_0, x_1, and x_2. Applying Simpson's rule again to integration with respect to x, from (4.46) we have

$$I \approx hk\left[f_{0,0} + f_{0,2} + f_{2,0} + f_{2,2} + 4\left\{f_{0,1} + f_{1,0} + f_{1,2} + f_{2,1}\right\} + 16f_{1,1}\right]/9 \tag{4.47}$$

where $h = x_2 - x_1 = x_1 - x_0$ and, for example, $f_{1,2} = f(x_1, y_2)$.

This is Simpson's rule in two variables. By applying this rule to each group of nine points on the surface $f(x,y)$ and summing, the composite Simpson's rule is obtained. The MATLAB function `simp2v` evaluates repeated integrals in two variables by making direct use of the composite rule.

```
function q = simp2v(func,a,b,c,d,n)
% Implements 2 variable Simpson integration.
% Example call: q = simp2v(func,a,b,c,d,n)
% Integrates user defined 2 variable function func.
% Range for first variable is a to b, and second variable, c to d
% using n divisions of each variable.
if (n/2)~=floor(n/2)
    disp('n must be even'); return
else
    hx = (b-a)/n; x = a:hx:b; nx = length(x);
    hy = (d-c)/n; y = c:hy:d; ny = length(y);
    [xx,yy] = meshgrid(x,y);
    z = feval(func,xx,yy);
    v = 2*ones(n+1,1);  v2 = 2*ones(n/2,1);
    v(2:2:n) = v(2:2:n)+v2;
    v(1) = 1;  v(n+1) = 1;
    S = v*v';  T = z.*S;
    q = sum(sum(T))*hx*hy/9;
end
```

We will now apply the function simp2v to evaluate the integral

$$\int_0^{10} dx \int_0^{10} y^2 \sin x \, dy$$

The graph of the function $y^2 \sin x$ is given in Figure 4.5. The following script integrates this function.

```
% e3s412.m
z = @(x,y) y.^2.*sin(x);
```

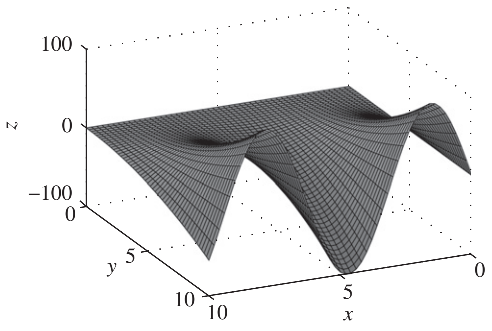

FIGURE 4.5 Graph of $z = y^2 \sin x$.

```
disp(' n     integral value');
n = 4; j = 1;
while n<=256
    int = simp2v(z,0,10,0,10,n);
    fprintf('%4.0f %17.8e\n',n,int)
    n = 2*n; j = j+1;
end
```

Running the script gives the following results:

```
n      integral value
  4    1.02333856e+003
  8    6.23187046e+002
 16    6.13568708e+002
 32    6.13056704e+002
 64    6.13025879e+002
128    6.13023970e+002
256    6.13023851e+002
```

The value of this integral exact to four decimal places is 613.0238 and the number of floating-point operations tends to $7n^2$. It can be proved—see Salvadori and Baron (1961)—that when Simpson's rule is adapted to evaluate repeated integrals, the error is still of order h^4 and thus it is possible to use an extrapolation scheme similar to the Romberg method of Section 4.6.

4.14.2 Gaussian Integration for Repeated Integrals

The Gaussian method can be developed to evaluate repeated integrals with constant limits of integration. In Section 4.7 it was shown that for single integrals the integrand must be evaluated at specified points. Thus, if

$$I = \int_{-1}^{1} dx \int_{-1}^{1} f(x,y)\, dy$$

then

$$I \approx \sum_{i=1}^{n} \sum_{j=1}^{m} A_i A_j f(x_i, y_j)$$

The rules for calculating x_i, y_j, and A_i are given in Section 4.7. The MATLAB function `gauss2v` evaluates integrals using this technique. Because the values of x and y are chosen on the assumption that the integration takes place in the range −1 to 1, the function includes the necessary manipulations to adjust it so as to accommodate an arbitrary range of integration.

```
function q = gauss2v(func,a,b,c,d,n)
% Implements 2 variable Gaussian integration.
% Example call: q = gauss2v(func,a,b,c,d,n)
% Integrates user defined 2 variable function func,
% Range for first variable is a to b, and second variable, c to d
% using n divisions of each variable.
% n must be 2 or 4 or 8 or 16.
if (n==2)|(n==4)|(n==8)|(n==16)
    co = zeros(8,4); t = zeros(8,4);
    co(1,1) = 1;
    co(1:2,2) = [.6521451548; .3478548451];
    co(1:4,3) = [.3626837833; .3137066458; .2223810344; .1012285362];
    co(:,4) = [.1894506104; .1826034150; .1691565193; .1495959888; ...
              .1246289712;.0951585116; .0622535239; .0271524594];
    t(1,1) = .5773502691;
    t(1:2,2) = [.3399810435; .8611363115];
    t(1:4,3) = [.1834346424; .5255324099; .7966664774; .9602898564];
    t(:,4) = [.0950125098; .2816035507; .4580167776; .6178762444; ...
             .7554044084; .8656312023; .9445750230; .9894009350];
    j = 1;
    while j<=4
        if 2^j==n; break;
        else
            j = j+1;
        end
    end
    s = 0;
    for k = 1:n/2
        x1 = (t(k,j)*(b-a)+a+b)/2;  x2 = (-t(k,j)*(b-a)+a+b)/2;
        for p = 1:n/2
            y1 = (t(p,j)*(d-c)+d+c)/2;  y2 = (-t(p,j)*(d-c)+d+c)/2;
            z = feval(func,x1,y1)+feval(func,x1,y2)+feval(func,x2,y1);
            z = z+feval(func,x2,y2);
            s = s+co(k,j)*co(p,j)*z;
        end
    end
    q = (b-a)*(d-c)*s/4;
else
    disp('n must be equal to 2, 4, 8 or 16'), return
end
```

We will now consider the problem of evaluating the following integral:

$$\int_{x^2}^{x^4} dy \int_{1}^{2} x^2 y \, dx \tag{4.48}$$

Integrals of this form cannot be estimated directly by the MATLAB function `gauss2v` or `simp2v` because neither of these functions was developed to work with variable limits of integration. However, a transformation may be carried out in order to make the limits of integration constant. Let

$$y = (x^4 - x^2)z + x^2 \tag{4.49}$$

Thus when $z = 1$, $y = x^4$ and when $z = 0$, $y = x^2$ as required. Differentiating the preceding expression, we have

$$dy = (x^4 - x^2)dz$$

Substituting for y and dy in (4.48), we have

$$\int_{0}^{1} dz \int_{1}^{2} x^2 \left\{(x^4 - x^2)z + x^2\right\} (x^4 - x^2) \, dx \tag{4.50}$$

This integral is now in a form that can be integrated using both `gauss2v` and `simp2v`. However, we must define a MATLAB function as follows:

```
w = @(x,z) x.^2.*((x.^4-x.^2).*z+x.^2).*(x.^4-x.^2);
```

This function is used with the functions `simp2v` and `gauss2v` in the following script:

```
% e3s413.m
disp('   n  Simpson value    Gauss value')
w = @(x,z) x.^2.*((x.^4-x.^2).*z+x.^2).*(x.^4-x.^2);
n = 2; j = 1;
while n<=16
    in1 = simp2v(w,1,2,0,1,n);
    in2 = gauss2v(w,1,2,0,1,n);
    fprintf('%4.0f%17.8e%17.8e\n',n,in1,in2)
    n = 2*n; j = j+1;
end
```

Running this script gives

```
  n  Simpson value    Gauss value
  2  9.54248047e+001  7.65255915e+001
  4  8.48837042e+001  8.39728717e+001
  8  8.40342951e+001  8.39740259e+001
 16  8.39778477e+001  8.39740259e+001
```

The integral is equal to 83.97402597 (= 6466/77). This output shows that, in general, Gaussian integration is superior to Simpson's rule.

4.15 MATLAB Functions for Double and Triple Integration

Recent versions of MATLAB now provide the functions `dblquad` and `triplequad` for repeated integration. In this section we consider these functions and their parameters and provide examples of their use.

For double integration, which is repeated integration over two dimensions, the `dblquad` function may be used and has the general form

```
IV2 = dblquad(fname,xl,xu,yl,yu,acc)
```

where `fname` is the name of the two-variable function being integrated, which must be defined by the user; `xl` and `xu` are the lower and upper limits of the x range of integration; and similarly `yl` and `yu` are the lower and upper limits for the y range of integration. The value `acc` provides the required accuracy of the integration and is optional.

The use of `dblquad` is illustrated by the following example. Consider the integral

$$I = \int_0^1 dx \int_0^1 \frac{1}{1-xy} dy$$

We may solve this using the MATLAB function `dblquad`. It is required that the user predefine the function to be integrated; to do this we choose to use an anonymous function directly in the function parameter list. Using `dblquad` we have

```
>> I = dblquad(@(x,y) 1./(1-x.*y),0,1-1e-6,0,1-1e-6)

I =
    1.6449
```

If we try to integrate this function numerically over the exact range $x = 0$ to 1 and $y = 0$ to 1 then MATLAB gives warnings because of the singularity when $x = y = 1$ but gives the same answer.

For triple integration, which is repeated integration over three dimensions, the `triplequad` function may be used and has the general form

```
IV3 = triplequad(fname,xl,xu,yl,yu,zl,zu,acc)
```

where `fname` is the name of the three variable function being integrated, and `xl` and `xu` are the lower and upper limits of the x range of integration. Similarly `yl`, `yu` and `zl`, `zu` are the limits for the y and z range of integration. The use of `triplequad` is illustrated by the following example:

$$\int_0^1 dx \int_0^1 dy \int_0^1 64xy(1-x)^2 z\, dz$$

```
>> I3 = triplequad(@(x,y,z) 64*x.*y.*(1-x).^2.*z,0,1,0,1,0,1)

I3 =
    1.3333
```

The function `quad2d` allows the user to integrate a function of two variables (say x and y) like the function `dblquad` but additionally allows the limits in y to be functions of x. Consider the integral (4.48) repeated here:

$$\int_1^2 dx \int_{x^2}^{x^4} x^2 y\, dy$$

Using `quad2d` we have

```
>> IV = quad2d(@(x,y) x.^2.*y,1,2, @(x) x.^2,@(x) x.^4)

IV =
   83.9740
```

In the preceding example, the anonymous function `@(x,y) x.^2.*y` is the function to be integrated, 1 and 2 are the lower and upper limits of integration in the `x` variable, and `@(x) x.^2` and `@(x) x.^4` are anonymous functions defining lower and upper limits in the `y` range of integration.

4.16 Summary

In this chapter we have described simple methods for obtaining the approximate derivatives of various orders for specified functions at given values of the independent variable. The results indicate that these methods, although easy to program, are very sensitive to small changes in key parameters and should be used with considerable care. In addition, we have given a range of methods for integration. For integration, error generation is not such an unpredictable problem but we must be careful to choose the most efficient method for the integral we wish to evaluate.

The reader is referred to Sections 9.8, 9.9, and 9.10 for the application of the Symbolic Toolbox to integration and differentiation problems.

Problems

4.1. Use the function `diffgen` to find the first and second derivatives of the function $x^2 \cos x$ at $x = 1$ using $h = 0.1$ and $h = 0.01$.

4.2. Evaluate the first derivative of $\cos x^6$ for $x = 1, 2$, and 3 using the function `diffgen` and taking $h = 0.001$.

4.3. Write a MATLAB function to differentiate a given function using formulae (4.6) and (4.7). Use it to solve Problems 4.1 and 4.2.

4.4. Find the gradient of $y = \cos x^6$ at $x = 3.1, 3.01, 3.001$, and 3 using the function `diffgen` with $h = 0.001$. Compare your results with the exact result.

4.5. The approximations for partial derivatives may be defined as

$$\partial f/\partial x \approx \{f(x+h, y) - f(x-h, y)\} / (2h)$$
$$\partial f/\partial y \approx \{f(x, y+h) - f(x, y-h)\} / (2h)$$

Write a function to evaluate these derivatives. The function call should have the form

```
[pdx,pdy] = pdiff('func',x,y,h)
```

Determine the partial derivatives of $\exp(x^2 + y^3)$ at $x = 2$, $y = 1$ using this function with $h = 0.005$.

4.6. In a letter sent to Hardy, the Indian mathematician Ramanujan proposed that the number of numbers between a and b that are either squares or sums of two squares is given approximately by the integral

$$0.764 \int_a^b \frac{dx}{\sqrt{\log_e x}}$$

Test this proposition for the following pairs of values of a and b: (1,10), (1,17), and (1,30). You should use the MATLAB function `fgauss` with 16 points to evaluate the integrals required.

4.7. Verify the equality

$$\int_0^\infty \frac{dx}{(1+x^2)(1+r^2x^2)(1+r^4x^2)} = \frac{\pi(r^2+r+1)}{2(r^2+1)(r+1)^2}$$

for the values of $r = 0, 1, 2$. This result was proposed by Ramanujan. You should use the MATLAB function `galag` for your investigations, using 8 points.

4.8. Raabe established the result that

$$\int_a^{a+1} \log_e \Gamma(x)dx = a\log_e a - a + \log_e \sqrt{2\pi}$$

Verify this result for $a = 1$ and $a = 2$. Use the MATLAB function `simp1` with 32 divisions to evaluate the integrals required and the MATLAB function `gamma` to set up the integrand.

4.9. Use the MATLAB function `fgauss` with 16 points to evaluate the integral

$$\int_0^1 \frac{\log_e x dx}{1+x^2}$$

Explain why the function `fgauss` is appropriate for this problem but `simp1` is not.

4.10. Use the MATLAB function `fgauss` with 16 points to evaluate the integral

$$\int_0^1 \frac{\tan^{-1} x}{x}\, dx$$

Note: Integration by parts shows the integrals in Problems 4.9 and 4.10 to be the same value except for a sign.

4.11. Write a MATLAB function to implement the formulae (4.32) and (4.33) given in Section 4.9 and use your function to evaluate the following integrals using 10 points for the formula. Compare your results with the Gauss 16-point rule.

$$\textbf{(a)} \int_{-1}^{1} \frac{e^x}{\sqrt{1-x^2}}\, dx \quad \textbf{(b)} \int_{-1}^{1} e^x\sqrt{1-x^2}\, dx$$

4.12. Use the MATLAB function `simp1` to evaluate the Fresnel integrals

$$C(1) = \int_0^1 \cos\left(\frac{\pi t^2}{2}\right) dt \quad \text{and} \quad S(1) = \int_0^1 \sin\left(\frac{\pi t^2}{2}\right) dt$$

Use 32 intervals. The exact values, to seven decimal places, are $C(1) = 0.7798934$ and $S(1) = 0.4382591$.

4.13. Use the MATLAB function `filon`, with 64 intervals, to evaluate the integral

$$\int_0^{\pi} \sin x \cos kx \, dx$$

for $k = 0$, 4, and 100. Compare your results with the exact answer, $2/(1-k^2)$ if k is even and 0 if k is odd.

4.14. Solve Problem 4.13 for $k = 100$ using Simpson's rule with 1024 divisions and Romberg's methods with 9 divisions.

4.15. Evaluate the following integral using the 8-point Gauss–Laguerre method:

$$\int_0^{\infty} \frac{e^{-x} dx}{x+100}$$

Compare your answer with the exact solution 9.9019419×10^{-3} (103/10402).

4.16. Evaluate the integral

$$\int_0^{\infty} \frac{e^{-2x} - e^{-x}}{x} dx$$

using 8-point Gauss–Lagurre integration. Compare your result with the exact answer, which is $-\log_e 2 = -0.6931$.

4.17. Evaluate the following integral using the 16-point Gauss–Hermite method. Compare your answer with the exact solution $\sqrt{\pi}\exp(-1/4)$.

$$\int_{-\infty}^{\infty} \exp(-x^2) \cos x \, dx$$

4.18. Evaluate the following integrals, using Simpson's rule for repeated integrals, MATLAB function `simp2v`, with 64 divisions in each direction.

$$\textbf{(a)} \int_{-1}^{1} dy \int_{-\pi}^{\pi} x^4 y^4 dx \quad \textbf{(b)} \int_{-1}^{1} dy \int_{-\pi}^{\pi} x^{10} y^{10} dx$$

4.19. Evaluate the following integrals, using `simp2v`, with 64 divisions in each direction.

$$(\mathbf{a}) \int_0^3 dx \int_1^{\sqrt{x/3}} \exp(y^3) dy \quad (\mathbf{b}) \int_0^2 dx \int_0^{2-x} (1+x+y)^{-3} dy$$

4.20. Evaluate part (b) in Problems 4.18 and 4.19 using Gaussian integration, MATLAB function `gauss2v`. *Note*: To use this function the range of integration must be constant.

4.21. The definition of the sine integral Si(z) is

$$\text{Si}(z) = \int_0^z \frac{\sin t}{t} dt$$

Evaluate this integral using the 16-point Gauss method for $z = 0.5, 1$, and 2. Why does the Gaussian method work and yet the Simpson and Romberg methods fail?

4.22. Evaluate the following double integral using Gaussian integration for two variables.

$$\int_0^1 dy \int_0^1 \frac{1}{1-xy} dx$$

Compare your result with the exact answer, $\pi^2/6 = 1.6449$.

4.23. The probability, P, that a certain type of gas turbine engine will fail within a period of time of T hours is given by the equation

$$P(x < T) = \int_0^T \frac{ab^a}{(x+b)^{a+1}} dx$$

where $a = 3.5$ and $b = 8200$.

By evaluating this integral for values of $T = 500 : 100 : 2000$, draw a graph of P against T in this range. What proportion of the number of gas turbines of this type fail within 1600 hours. For more information on the probability of failure, see Percy (2011).

4.24. Consider the following integral:

$$\int_0^1 \frac{x^p - x^q}{\log_e(x)} x^r dx = \log_e\left(\frac{p+r+1}{q+r+1}\right)$$

Use the MATLAB function `quad` to verify this result for $p = 3, q = 4, r = 2$.

4.25. Consider the following three integrals:

$$A = -\int_0^1 \frac{\log_e x\, dx}{1+x^2}, \quad B = \int_0^1 \frac{\tan^{-1} x}{x} dx, \quad C = \int_0^\infty \frac{xe^{-x}}{1+e^{-2x}} dx$$

Use the MATLAB function `quad` to evaluate the two integrals A and B and hence verify that they are equal.

Use 8-point Gauss–Laguerre integration to verify that the integral C is also equal to A and B.

4.26. Use 16-point Gauss–Hermite integration to evaluate the integral

$$I = \int_{-\infty}^{\infty} \frac{\sin x}{1+x^2} dx$$

and show that its value is approximately equal to zero.

4.27. Use 16-point Gauss–Hermite integration to evaluate the integral

$$I = \int_{-\infty}^{\infty} \frac{\cos x}{1+x^2} dx$$

Check your answer by comparing with the exact answer, π/e.

4.28. Use 8-point Gauss–Lagurre integration to find the value of the integral

$$I = \int_0^\infty \frac{x^{\alpha-1}}{1+x^\beta} dx$$

for values of α and $\beta = (2,3)$, $(3,4)$. You can verify your answers using the exact value of the integral, which is $\pi/(\beta \sin(\alpha\pi/\beta))$.

4.29. An interesting relationship between the Riemann zeta function and the integral

$$S_3 = -\int_0^\infty \log_e(x)^3 e^{-x}\, dx$$

is given by

$$S_3 = \gamma^3 + \frac{1}{2}\gamma\pi^2 + 2\zeta(3)$$

where $\gamma = 0.57722$. Use the MATLAB function `quadgk` to evaluate the integral and show that it is a good estimate of S_3.

4.30. A value for the total resistance of a certain network of unit resistors has been shown to be given by $R(m, n)$, where

$$R(m,n) = \frac{1}{\pi^2} \int_0^\pi dx \int_0^\pi \frac{1 - \cos mx \cos ny}{2 - \cos x - \cos y}\, dy$$

Evaluate this integral for $R(50, 100)$ using the MATLAB functions `dblquad` and `simp2v`. Use a lower limit close to zero, say 0.0001. If zero is used the denominator in the integral is zero. For large values of m and n an approximation for this integral is given by

$$R(m,n) = \frac{1}{\pi}\left(\gamma + \frac{3}{2}\log_e 2 + \frac{1}{2}\log_e(m^2 + n^2)\right)$$

where γ is Euler's constant and can be obtained by evaluating the MATLAB expression `-psi(1)`. The function `-psi` is called the digamma function. Use this to check your result.

4.31. A value for the total resistance of a cubic network of unit resistors has been shown to be given by $R(s, m, n)$ where

$$R(s,m,n) = \frac{1}{\pi^3} \int_0^\pi dx \int_0^\pi dy \int_0^\pi \frac{1 - \cos sx \cos my \cos nz}{3 - \cos x - \cos y - \cos z}\, dz$$

Evaluate this integral using the MATLAB function for `triplequad` using the values $s = 2$, $m = 1$, $n = 3$. The lower limit should be set at a small nonzero value, say 0.0001.