9 Applications of the Symbolic Toolbox

The Symbolic Toolbox provides an extensive list of functions for the symbolic manipulation of symbolic expressions and equations. The use of symbolic functions and their analytic manipulation can often play a useful role in association with a numerical algorithm. In such algorithms the combination of the standard numerical functions with the facilities of the Symbolic Toolbox can be particularly beneficial, relieving the user of the tedious and error-prone task of symbolic manipulation. This allows the designer of the algorithm to provide a more user-friendly and complete function.

Originally the MATLAB Symbolic Toolbox used the Maplesoft symbolic software to carry out symbolic operations and pass the results to MATLAB. However, since late 2008 MATLAB has used Mupad for its symbolic engine. This change has caused minor changes in the way results are presented, but the results are generally equivalent.

9.1 Introduction to the Symbolic Toolbox

Since we are using the Symbolic Toolbox in the field of numerical analysis we begin by giving some examples of the beneficial application of this toolbox. It provides

1. The symbolic first derivative of a given single-variable nonlinear function, which is required by Newton's method for the solution of single-variable nonlinear equations (see Chapter 3)
2. The Jacobian for a system of nonlinear simultaneous equations (see Chapter 3)
3. The symbolic gradient vector of a given nonlinear function, which is required for the conjugate gradient method for minimizing a nonlinear function (see Chapter 8)

An important feature of the Symbolic Toolbox is that it allows an extra dimension of experimentation. For example, a user can test a given numerical algorithm by solving a test problem symbolically, providing it has a solution in the closed form, and compare this exact solution with a numerical solution. In addition, the study of the accuracy of computations can be enhanced using the Symbolic Toolbox variable-precision arithmetic feature. This feature allows the user to perform certain computations to an unlimited precision.

In Sections 9.2 through 9.14 we provide an introduction to some of the Symbolic Toolbox features but it is not our intention to provide details of all the features. In Section 9.15 we describe applications of the Symbolic Toolbox to specific numerical algorithms.

Numerical Methods Using MATLAB®. DOI: 10.1016/B978-0-12-386942-5.00009-6

9.2 Symbolic Variables and Expressions

The first key point to note is that symbolic variables and expressions are different from the standard variables and expressions of MATLAB, and we must distinguish clearly between them. Symbolic variables and symbolic expressions do not have to have numerical values but rather define a structural relationship between symbolic variables, that is, an algebraic expresssion.

To define any variable as a symbolic variable the `sym` function must be used as follows:

```
>> x = sym('x')

x =
x

>> d1 = sym('d1')

d1 =
d1
```

Alternatively, we can use the statement `syms` to define any number of symbolic variables. Thus

```
>> syms a b c d3
```

provides four symbolic variables `a`, `b`, `c`, and `d3`. Note that there is no output to the screen. This is a useful shortcut for the definition of variables and we have used this approach in this text. To check which variables have been declared as symbolic we can use the standard `whos` command. Thus if we use this command after the preceding `syms` declaration, we obtain

```
>> whos
  Name      Size            Bytes  Class    Attributes

  a         1x1                60  sym
  ans       1x19               38  char
  b         1x1                60  sym
  c         1x1                60  sym
  d1        1x1                60  sym
  d3        1x1                60  sym
  x         1x1                60  sym
```

Once variables have been defined as symbolic, expressions can be written using them directly in MATLAB and they will be treated as symbolic expressions. For example, once `x` has been defined as a symbolic variable, the statement

```
>> syms x
>> 1/(1+x)
```

produces the symbolic expression

```
ans =
1/(x + 1)
```

To set up a symbolic matrix we first define any symbolic variables involved in the matrix. Then we enter the statement that defines the matrix in terms of these symbolic variables in the usual manner. On execution the matrix will be displayed as follows:

```
>> syms x y
>> d = [x+1 x^2 x-y;1/x 3*y/x 1/(1+x);2-x x/4 3/2]

d =
[ x + 1,     x^2,     x - y]
[   1/x, (3*y)/x, 1/(x + 1)]
[ 2 - x,     x/4,       3/2]
```

Note that `d` is automatically made symbolic by the assignment of a symbolic expression. We can address individual elements or specific rows and columns as follows:

```
>> d(2,2)

ans =
(3*y)/x

>> c = d(2,:)

c =
[ 1/x, (3*y)/x, 1/(x + 1)]
```

We now consider the manipulation of a symbolic expression. First we set up a symbolic expression as follows:

```
>> e = (1+x)^4/(1+x^2)+4/(1+x^2)

e =
(x + 1)^4/(x^2 + 1) + 4/(x^2 + 1)
```

To see more clearly what this expression represents we can use the function `pretty` to get a more conventional layout of the function:

```
>> pretty(e)

         4
  (x + 1)       4
  -------- + ------
    2          2
   x  + 1     x  + 1
```

Not very pretty! We can simplify the symbolic expression `e` using the function `simplify`:

```
>> simplify(e)

ans =
x^2 + 4*x + 5
```

We can expand expressions using `expand`:

```
>> p = expand((1+x)^4)

p =
x^4 + 4*x^3 + 6*x^2 + 4*x + 1
```

Note the layout of this and other expressions may vary slightly from one computer platform to another. The expression for `p`, in turn, can be rearranged into a nested form using the function `horner`:

```
>> horner(p)

ans =
x*(x*(x*(x + 4) + 6) + 4) + 1
```

We may factorize expressions using the function `factor`. Assuming `a`, `b`, and `c` have been declared as symbolic variables, then

```
>> syms a b c
>> factor(a^3+b^3+c^3-3*a*b*c)

ans =
(a + b + c)*(a^2 - a*b - a*c + b^2 - b*c + c^2)
```

When dealing with complicated expressions, it is useful to simplify the expression as far as possible and as soon as possible. However, it is not always immediately obvious which route should be taken in the simplification process. The function `simple` attempts to simplify the expression using a variety of methods and displays the various results to inform the user. Some of the methods used by the function are not available as separate functions, for example, `radsimp` and `combine(trig)`. The following illustrates the use of the function `simple`:

```
>> syms x; y = sqrt(cos(x)+i*sin(x));
>> simple(y)

simplify:
(cos(x) + sin(x)*i)^(1/2)

radsimp:
(cos(x) + sin(x)*i)^(1/2)

simplify(100):
exp(x*i)^(1/2)

.................
.................

rewrite(exp):
exp(x*i)^(1/2)

rewrite(sincos):
(cos(x) + sin(x)*i)^(1/2)

rewrite(sinhcosh):
(cosh(x*i) + sinh(x*i))^(1/2)

rewrite(tan):
((tan(x/2)*2*i)/(tan(x/2)^2 + 1)
                 - (tan(x/2)^2 - 1)/(tan(x/2)^2 + 1))^(1/2)

mwcos2sin:
(sin(x)*i - 2*sin(x/2)^2 + 1)^(1/2)

collect(x):
(cos(x) + sin(x)*i)^(1/2)

ans =
exp(x*i)^(1/2)
```

In this example the symbolic engine has tried no fewer than fifteen methods to simplify the original expression (not all displayed here)—many to little effect. However, the multiplicity of methods provided allows problems of differing algebraic and transcendental functions to be simplified. The final answer is the shortest, and we would judge it the most acceptable. A compact version of this result can be determined using

```
>> [r,how] = simple(y)

r =
exp(x*i)^(1/2)

how =
simplify(100)
```

When manipulating algebraic, trigonometric, and other expressions, it is important to be able to substitute an expression or constant for any given variable. For example,

```
>> syms u v w
>> fmv = pi*v*w/(u+v+w)

fmv =
(pi*v*w)/(u + v + w)
```

Now we substitute for various variables in this expression. The following statement substitutes the symbolic expression `2*v` for the variable `u`:

```
>> subs(fmv,u,2*v)

ans =
(pi*v*w)/(3*v + w)
```

This next statement substitutes the symbolic constant `1` for the variable `v` in the previous result held in `ans`:

```
>> subs(ans,v,1)

ans =
(pi*w)/(w + 3)
```

Finally, we substitute the symbolic constant `1` for `w` to give

```
>> subs(ans,w,1)

ans =
    0.7854
```

As a further example of the use of the subs function, consider the statements

```
>> syms y
>> f = 8019+20412*y+22842*y^2+14688*y^3+5940*y^4 ...
                                     +1548*y^5+254*y^6+24*y^7+y^8;
```

Now to substitute x-3 for y we use

```
>> subs(f,y,x-3)

ans =
20412*x + 22842*(x - 3)^2 + 14688*(x - 3)^3 + 5940*(x - 3)^4
   + 1548*(x - 3)^5 + 254*(x - 3)^6 + 24*(x - 3)^7 + (x - 3)^8 - 53217
```

Using the collect function to collect terms having the same power of x, we obtain a major simplification as follows:

```
>> collect(ans)

ans =
x^8 + 2*x^6
```

The process of rearranging and simplifying algebraic and transcendental expressions is difficult and the Symbolic Toolbox can be both powerful and frustrating. Powerful because, as we have shown, it is capable of simplifying complex expressions; frustrating because it sometimes fails on relatively simple problems.

Now that we have seen how to manipulate symbolic expressions, we may require a graphical representation. A simple approach to plotting symbolic functions is to use the MATLAB function ezplot, although it must be emphasized that this is restricted to single-variable functions. The following function provides the plot of the normal curve between the values −5 and 5, as shown in Figure 9.1.

```
>> syms x
>> ezplot(exp(-x*x/2),-5,5); grid
```

The alternative to using ezplot is to substitute numerical values into the symbolic expression using the subs function and then use conventional plotting functions.

9.3 Variable-Precision Arithmetic in Symbolic Calculations

In symbolic calculations involving numerical values, the function vpa may be used to obtain any number of decimal places. It should be noted that the result of using this function is a symbolic constant, not a numerical value. Thus to provide $\sqrt{6}$ to 100 places we write vpa(sqrt(6),100). The accuracy here is not restricted to 16 decimal places as in

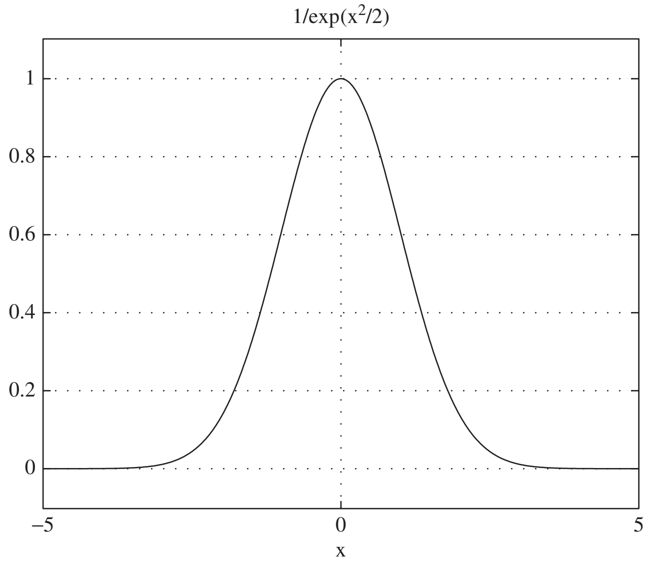

FIGURE 9.1 A plot of the normal curve using the function `ezplot`.

ordinary arithmetic calculations. A nice illustration of this feature is given in the following where we provide an implementation of the famous algorithm of the Borweins, which amazingly quadruples the number of accurate decimal places of π at each iteration! We include this script for illustration only; MATLAB can give π to as many digits as required by writing, for example, `vpa(pi,100)`.

```
% Script e3s901.m  Borwein iteration for pi
n = input('enter n')
y0 = sqrt(2)-1; a0 = 6-4*sqrt(2);
np = 4;
for k = 0:n
    yv = (1-y0^4)^0.25; y1 = (1-yv)/(1+yv);
    a1 = a0*(1+y1)^4-2.0^(2*k+3)*y1*(1+y1+y1^2);
    rpval = a1;  pval = vpa(1/rpval,np)
    a0 = a1; y0 = y1; np = 4*np;
end
```

The results of three iterations follow:

```
enter n 3

n =
     3

pval =
3.142
```

```
pval =
3.141592653589793

pval =
3.1415926535897932384626433832795028841971693993751058209749445922

pval =
3.1415926535897932384626433832795028841971693993751058209749445922
3078164062862089986280348253421170679821480865132823066470938446
0955058223172535940812848111745028410270193852110555964462294895
4930381964428810975665933446128475648233786783165271201909145649
```

Theoretically, the function vpa can be utilized to give results to any number of decimal places. An excellent introduction to the computation of π is given by Bailey (1988).

9.4 Series Expansion and Summation

In this section we consider both the development of series approximations for functions and the summation of series.

We begin by showing how a symbolic function can be expanded in the form of a Taylor series using the MATLAB function taylor(f,n). This provides the $(n-1)$th-degree polynomial approximation for the symbolically defined function f. If the taylor function has only one parameter, then it provides the fifth-degree polynomial approximation for that function.

Consider the following examples:

```
>> syms x
>> taylor(cos(exp(x)),4)

ans =
- (cos(1)*x^3)/2 + (- cos(1)/2 - sin(1)/2)*x^2 - sin(1)*x + cos(1)

>> s = taylor(exp(x),8)

s =
x^7/5040 + x^6/720 + x^5/120 + x^4/24 + x^3/6 + x^2/2 + x + 1
```

The series expansion for the exponential function can be summed using the function symsum with $x = 0.1$. To use this function we must know the form of the general term. In this case the series definition is

$$e^{0.1} = \sum_{r=1}^{\infty} \frac{0.1^{r-1}}{(r-1)!} \quad \text{or} \quad \sum_{r=1}^{\infty} \frac{0.1^{r-1}}{\Gamma(r)}$$

Thus, to sum the first eight terms we have

```
>> syms r
>> symsum((0.1)^(r-1)/gamma(r),1,8)

ans =
55700614271/50400000000
```

This can then be evaluated using the function `double`:

```
>> double(ans)

ans =
    1.1052
```

In this example a simple alternative is to use the function `subs` to replace `x` by 0.1 in the symbolic function represented by `s`, and then use `double` to evaluate it, as follows:

```
>> double(subs(s,x,0.1))

ans =
    1.1052
```

This gives a good approximation since we can see that

```
>> exp(0.1)

ans =
    1.1052
```

The function `symsum` can be used to perform many different summations using different combinations of parameters. The following examples illustrate the different cases. To sum the series

$$S = 1 + 2^2 + 3^2 + 4^2 + \cdots + n^2 \tag{9.1}$$

we proceed as follows:

```
>> syms r n
>> symsum(r*r,1,n)

ans =
(n*(2*n + 1)*(n + 1))/6
```

Another example is to sum the series

$$S = 1 + 2^3 + 3^3 + 4^3 + \cdots + n^3 \tag{9.2}$$

Here we use

```
>> symsum(r^3,1,n)

ans =
(n^2*(n + 1)^2)/4
```

Infinity can be used as an upper limit. As an example of this, consider the following infinite sum:

$$S = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots \frac{1}{r^2} + \cdots$$

Summing an infinity of terms, we have

```
>> symsum(1/r^2,1,inf)

ans =
pi^2/6
```

This is an interesting series and is a particular case of the Riemann zeta function (implemented in MATLAB by `zeta(k)`), which gives the sum of the following series:

$$\zeta(k) = 1 + \frac{1}{2^k} + \frac{1}{3^k} + \frac{1}{4^k} + \cdots + \frac{1}{r^k} + \cdots \tag{9.3}$$

For example:

```
>> zeta(2)

ans =
    1.6449

>> zeta(3)

ans =
    1.2021
```

A further interesting example is a summation involving the gamma function (Γ), where $\Gamma(r) = 1.2.3\ldots(r-2)(r-1) = (r-1)!$ for integer values of r. This function is implemented in MATLAB by `gamma(r)`. For example, to sum the series

$$S = 1 + \frac{1}{1} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{r!} + \cdots$$

to infinity, we use the MATLAB statement

```
>> symsum(1/gamma(r),1,inf)

ans =
exp(1)

>> vpa(ans,100)

ans =
2.718281828459045235360287471352662497757247093699959574966967627
7240766303535475945713821785251664274
```

Notice that the use of the `vpa` function leads to an interesting evaluation of e to a large number of decimal places.

A further example is given by

$$S = 1 + \frac{1}{1!} + \frac{1}{(2!)^2} + \frac{1}{(3!)^2} + \frac{1}{(4!)^2} + \cdots$$

In MATLAB, this becomes

```
>> symsum(1/gamma(r)^2,1,inf)

ans =
sum(1/gamma(r)^2, r = 1..Inf)
```

This is an example of a case where `symsum` has not worked.

9.5 Manipulation of Symbolic Matrices

Some of the MATLAB functions, such as `eig`, that can be applied to numerical matrices can also be applied directly to symbolic matrices. However, these features must be used with care for two reasons. First, manipulating large symbolic matrices can be a very slow process. Second, the symbolic results derived from such operations can be of such algebraic complexity that it is difficult or almost impossible to obtain insight into the meaning of the equation.

We will begin by finding the eigenvalues of a simple 4×4 matrix expressed in terms of two symbolic variables.

```
>> syms a b
>> Sm = [a b 0 0;b a b 0;0 b a b;0 0 b a]

Sm =
[ a, b, 0, 0]
[ b, a, b, 0]
[ 0, b, a, b]
[ 0, 0, b, a]
```

```
>> eig(Sm)

ans =
 a - b/2 - (5^(1/2)*b)/2
 a - b/2 + (5^(1/2)*b)/2
 a + b/2 - (5^(1/2)*b)/2
 a + b/2 + (5^(1/2)*b)/2
```

In this problem the expressions for the eigenvalues are quite simple. In contrast, we will now consider an example that appears to be equally simple but develops into a problem with eigenvalues that are much more complicated.

```
>> syms A p
>> A = [1 2 3;4 5 6;5 7 9+p]

A =
[ 1, 2,     3]
[ 4, 5,     6]
[ 5, 7, p + 9]
```

Inspection of this matrix reveals that if $p = 0$, the matrix is singular. Evaluating the determinant of the matrix, we have

```
>> det(A)

ans =
-3*p
```

From this simple result it can immediately be seen that as p tends to zero, the determinant of the matrix tends to zero, indicating that the matrix is singular. Inverting the matrix gives

```
>> B = inv(A)

B =
[    -(5*p + 3)/(3*p), (2*p - 3)/(3*p),  1/p]
[ (2*(2*p + 3))/(3*p),  -(p - 6)/(3*p), -2/p]
[                -1/p,            -1/p,  1/p]
```

This is much more difficult to interpret, although it can be seen that as p tends to zero, each element of the inverse matrix tends to infinity; that is, the inverse does not exist. Note that every element of this inverse matrix is a function of p whereas only one element of the original matrix is a function of p. Finally, we can compute the eigenvalues of the original matrix using the statement `v = eig(A)`. The value of the symbolic object `v` is not shown here because it is so long and complicated. We can find out how many characters (including spaces) are required to express the three eigenvalues symbolically as follows:

```
>>n = length(char(v))

n =
1720
```

This output is very difficult to read, let alone understand. Using the pretty print facility (`pretty`) improves the situation, but the ouput still requires 106 character spaces per line—far more than can be displayed on this page.

The following scripts compute the eigenvalues both symbolically and numerically. In each case the parameter p is varied from 0 to 2 in steps of 0.1. However, only the eigenvalues that correspond to $p = 0.9$ and 1.9 are displayed. The following script determines the eigenvalues symbolically, and then the values of p are substituted into the symbolic eigenvalue expressions using the function `subs`; the function `double` is then used to provide the numerical results.

```
% e3s902.m
disp('Script 1; Symbolic - numerical solution')
c = 1; v = zeros(3,21);
tic
syms a p u w
a = [1 2 3;4 5 6;5 7 9+p];
w = eig(a);  u = [ ];
for s = 0:0.1:2
    u = [u,subs(w,p,s)];
end
v = sort(real(double(u)));
toc
v(:,[10 20])
```

Running this script gives

```
Script 1; Symbolic - numerical solution
Elapsed time is 2.108940 seconds.

ans =
   -0.4255   -0.4384
    0.3984    0.7854
   15.9270   16.5530
```

An alternative approach is to find the eigenvalues of the same matrix by substituting numerical values of p into the numerical matrix and thus determine the eigenvalues. A script to carry out this process follows. Again, only the eigenvalues for $p = 0.9$ and 1.9 are displayed.

```
% e3s903
disp('Script 2: Numerical solution')
c = 1; v = zeros(3,21);
tic
for p = 0:.1:2
    a = [1 2 3;4 5 6;5 7 9+p];
    v(:,c) = sort(eig(a));
    c = c+1;
end
toc
v(:,[10 20])
```

Running this script gives

```
Script 2: Numerical solution
Elapsed time is 0.000934 seconds.

ans =
   -0.4255   -0.4384
    0.3984    0.7854
   15.9270   16.5530
```

As expected, the methods give identical results and show that the eigenvalues are real. Note that the symbolic approach is much slower.

We conclude this section with an example that illustrates the advantage of the symbolic approach for certain problems. We wish to find the eigenvalues of a matrix that can be generated using the MATLAB statement `gallery(5)`. We will begin by finding the eigenvalues of this matrix in a nonsymbolic way.

```
>> B = gallery(5)

B =
          -9          11         -21          63        -252
          70         -69         141        -421        1684
        -575         575       -1149        3451      -13801
        3891       -3891        7782      -23345       93365
        1024       -1024        2048       -6144       24572
```

```
>> format long e
>> eig(B)

ans =
    -4.052036755439267e-002
    -1.177933343414123e-002 +3.828611372186529e-002i
    -1.177933343414123e-002 -3.828611372186529e-002i
     3.203951721060507e-002 +2.281159217067240e-002i
     3.203951721060507e-002 -2.281159217067240e-002i
```

The eigenvalues appear to be small with one real value and the remaining values forming complex conjugate pairs. However, using the symbolic approach we have

```
>> A = sym(gallery(5))

A =
[   -9,    11,   -21,     63,   -252]
[   70,   -69,   141,   -421,   1684]
[ -575,   575, -1149,   3451, -13801]
[ 3891, -3891,  7782, -23345,  93365]
[ 1024, -1024,  2048,  -6144,  24572]

>> eig(A)

ans =
 0
 0
 0
 0
 0
```

How do we verify which of these two solutions is correct? If we rearrange the eigenvalue problem into the form given by (2.38), that is,

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{x} = 0$$

then the eigenvalues are the roots of $|\mathbf{A} - \lambda\mathbf{I}| = 0$. We can find these roots symbolically in MATLAB as follows:

```
>> syms lambda
>> D = A-lambda*sym(eye(5));
>> det(D)

ans =
-lambda^5
```

We have shown that $|\mathbf{A} - \lambda\mathbf{I}| = -\lambda^5$ and hence the eigenvalues are the roots of $-\lambda^5 = 0$, that is, zero. Here the Symbolic Toolbox has revealed the true solution.

9.6 Symbolic Methods for the Solution of Equations

The function available in MATLAB to solve symbolic equations is `solve`. This function is most useful for solving polynomials since it provides expressions for all roots. To use `solve` we must set up the expression for the equation we wish to solve in terms of a symbolic variable. For example,

```
>> syms x
>> f = x^3-7/2*x^2-17/2*x+5

f =
x^3 - (7*x^2)/2 - (17*x)/2 + 5

>> solve(f)

ans =
   5
  -2
 1/2
```

The following example illustrates how to solve a system of two equations in two variables. In this example we have chosen to enter the two equations directly in the function by placing them in quotes.

```
>> syms x y
>> [x y] = solve('x^2+y^2=a','x^2-y^2=b')
```

This gives four solutions:

```
x =
  (2^(1/2)*(a + b)^(1/2))/2
 -(2^(1/2)*(a + b)^(1/2))/2
  (2^(1/2)*(a + b)^(1/2))/2
 -(2^(1/2)*(a + b)^(1/2))/2

y =
  (2^(1/2)*(a - b)^(1/2))/2
  (2^(1/2)*(a - b)^(1/2))/2
 -(2^(1/2)*(a - b)^(1/2))/2
 -(2^(1/2)*(a - b)^(1/2))/2
```

A simple check of this solution may be obtained by inserting these solutions back in the original equations:

```
>> x.^2+y.^2, x.^2-y.^2

ans =
 a
 a
 a
 a

ans =
 b
 b
 b
 b
```

Hence all four solutions satisfy the equations simultaneously.

It should be noted that if `solve` fails to obtain a symbolic solution to a given equation or set of equations, it will attempt to use the standard numerical processes where appropriate. In practice it is rarely possible to determine the symbolic solutions of general single-variable or multivariable nonlinear equations.

9.7 Special Functions

The MATLAB Symbolic Toolbox provides the user with access to a range of over 50 special functions and polynomials that can be used symbolically. These functions are not m-files and the standard MATLAB `help` command cannot be used to gain information about them. We can obtain a list of these functions using the command `help mfunlist`. The function `mfun` allows these functions to be evaluated numerically.

One of these functions is the Fresnel sine integral. In `mfunlist` the function `FresnelS` defines the Fresnel sine integral of x. To evaluate it for $x = 4.2$ we enter

```
>> x = 4.2; y = mfun('FresnelS',x)

y =
    0.5632
```

Note that the first and last letters in `FresnelS` must be capitalized. We plot this function (Figure 9.2) using the following script:

```
>> x=1:.01:3; y = mfun('FresnelS',x);
>> plot(x,y)
>> xlabel('x'), ylabel('Fresnel sine integral')
```

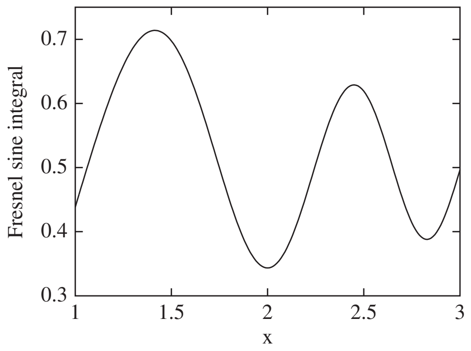

FIGURE 9.2 Plot of the Fresnel sine integral.

Another interesting function available in the `mfunlist` is the logarithmic integral, Li. For example,

```
>> y = round(mfun('Li',[1000 10000 100000]))

y =
          178        1246        9630
```

The logarithmic integral can be used to predict the number of primes below a particular value, and the prediction becomes more accurate as the number of primes increases. To find the number of primes below 1000, 10000, and 100000 we have

```
>> p1 = length(primes(1000)); p2 = length(primes(10000));
>> p3 = length(primes(100000)); p = [p1 p2 p3]

p =
          168        1229        9592
```

Taking the ratio between the values of the logarithmic integral `y` and the number of primes `p`, we have

```
>> p./y

ans =
     0.9438    0.9864    0.9961
```

Notice that the ratio of the value of the logarithmic integral and the number of primes less N tends to 1 as N tends to infinity.

Two important functions are the Dirac delta and Heaviside functions. These functions are not part of the `mfunlist`, and information about them can be obtained using the command `help` in the usual way.

The Dirac delta, or impulse, function $\delta(x)$, is defined as follows:

$$\delta(x - x_0) = 0 \quad x \neq x_0 \tag{9.4}$$

$$\int_{-\infty}^{\infty} f(x)\,\delta(x - x_0)\,dx = f(x_0) \tag{9.5}$$

If $f(x) = 1$, then

$$\int_{-\infty}^{\infty} \delta(x - x_0)\,dx = 1 \tag{9.6}$$

By (9.4), the Dirac delta function only exists at $x = x_0$. Furthermore, from (9.6), the area under the Dirac delta function is unity. This function is implemented in MATLAB by `dirac(x)`.

The Heaviside, or unit step, function is defined as follows:

$$u(x - x_0) = \begin{cases} 1, & \text{for} \quad x - x_0 > 0 \\ 0.5, & \text{for} \quad x - x_0 = 0 \\ 0, & \text{for} \quad x - x_0 < 0 \end{cases} \tag{9.7}$$

This function is implemented in MATLAB by `heaviside(x)`. Thus

```
heaviside(2)

ans =
     1
```

In Sections 9.8, 9.10, and 9.14, we illustrate some of the properties of these functions.

9.8 Symbolic Differentiation

Essentially any function can be differentiated symbolically, but the process is not always easy. For example, the following function given by Swift (1977) is tedious to differentiate:

$$f(x) = \sin^{-1}\left(\frac{e^x \tan x}{\sqrt{x^2 + 4}}\right)$$

Differentiating this function using the MATLAB function `diff` gives

```
>> syms x
>> diff(asin(exp(x)*tan(x)/sqrt(x^2+4)))

ans =
((exp(x)*(tan(x)^2 + 1))/(x^2 + 4)^(1/2) +
(exp(x)*tan(x))/(x^2 + 4)^(1/2) -
(x*exp(x)*tan(x))/(x^2 + 4)^(3/2))/
(1 - (exp(2*x)*tan(x)^2)/(x^2 + 4))^(1/2)

>> pretty(ans)
                2
  exp(x) (tan(x)  + 1)   exp(x) tan(x)   x exp(x) tan(x)
  -------------------- + ------------- - ---------------
        2     1/2           2     1/2        2     3/2
      (x  + 4)            (x  + 4)         (x  + 4)
  -------------------------------------------------------
               /                    2 \1/2
               |     exp(2 x) tan(x)  |
               | 1 - ---------------- |
               |           2          |
               \          x  + 4      /
```

We now explain specifically how this is achieved. Differentiation is performed on a specified function with respect to a specific variable. When using the Symbolic Toolbox, the variables or variable of the expression must be defined as symbolic. Once this is done, the expression will be treated as symbolic and differentiation may be performed. Consider the following example:

```
>> syms z k
>> f = k*cos(z^4);
>> diff(f,z)

ans =
-4*k*z^3*sin(z^4)
```

Note that the differentiation is performed with respect to `z`, the variable indicated in the second parameter. Differentiation may be performed with respect to `k` as follows:

```
>> diff(f,k)

ans =
cos(z^4)
```

If the variable with respect to which the differentiation is being performed is not indicated, MATLAB chooses the variable name alphabetically closest to x.

Higher-order differentiation can also be performed by including an additional integer parameter that indicates the order of the differentiation as follows:

```
>> syms n
>> diff(k*z^n,4)

ans =
k*n*z^(n - 4)*(n - 1)*(n - 2)*(n - 3)
```

The following example illustrates how we can obtain a standard numerical value from our symbolic differentiation:

```
>> syms x
>> f = x^2*cos(x);
>> df = diff(f)

df =
2*x*cos(x) - x^2*sin(x)
```

We can now substitute a numerical value for `x`:

```
>> subs(df,x,0.5)

ans =
    0.7577
```

Finally, we consider the symbolic differentiation of the Heaviside, or unit step, function:

```
 diff(heaviside(x))

ans =
dirac(x)
```

This is as expected since the differention of the Heaviside function is zero for all x except when $x = 0$.

9.9 Symbolic Partial Differentiation

Partial derivatives of any multivariable function can be found by differentiating with respect to each variable in turn. As an example, we set up a symbolic function of three variables, assigned to fmv, as follows:

```
>> syms u v w
>> fmv =u*v*w/(u+v+w)

fmv =
(u*v*w)/(u + v + w)

>> pretty(fmv)
    u v w
  ---------
  u + v + w
```

Now we differentiate with respect to u, v, and w in turn:

```
>> d = [diff(fmv,u) diff(fmv,v) diff(fmv,w)]

d =
[ (v*w)/(u + v + w) - (u*v*w)/(u + v + w)^2,
      (u*w)/(u + v + w) - (u*v*w)/(u + v + w)^2,
                (u*v)/(u + v + w) - (u*v*w)/(u + v + w)^2]
```

To obtain mixed partial derivatives we simply differentiate with respect to the second variable the answer obtained after differentiation with respect to the first. For example,

```
>> diff(d(3),u)

ans =
v/(u + v + w) - (u*v)/(u + v + w)^2
                  - (v*w)/(u + v + w)^2 + (2*u*v*w)/(u + v + w)^3
```

To clarify the structure of the preceding expression, we use the function pretty:

```
>> pretty(ans)
      v           u v            v w          2 u v w
  --------- - ------------ - ------------ + ------------
  u + v + w              2              2              3
              (u + v + w)    (u + v + w)    (u + v + w)
```

This expression provides the mixed second-order partial derivative with respect to w and then u.

9.10 Symbolic Integration

The integration process presents more difficulties than differentiation because not all functions can be integrated in the closed form to produce a symbolic algebraic expression. Even functions that can be integrated in the closed form often require considerable skill and experience for their evaluation.

To use the Symbolic Toolbox we begin by defining the symbolic expression `f`. Then the function `int(f,a,b)` performs symbolic integration where `a` and `b` are the lower and upper limits, respectively. This results in a symbolic constant. If the upper and lower limits are omitted, the result is an expression that is the formula for the indefinite integral. In either case, if the integral cannot be evaluated, which frequently happens, the original function is returned. It must be stressed that many integrals can only be evaluated numerically.

Consider the following indefinite integral:

$$I = \int u^2 \cos u \, du$$

In MATLAB this becomes

```
>> syms u
>> f = u^2*cos(u); int(f)

ans =
sin(u)*(u^2 - 2) + 2*u*cos(u)
```

We note that the result is a formula as expected and not a numerical value. However, if upper and lower limits are specified, we obtain a symbolic constant. For example, consider

$$y = \int_0^{2\pi} e^{-x/2} \cos(100x) dx$$

Thus we have

```
>> syms x, res = int(exp(-x/2)*cos(100*x),0,2*pi)

res =
2/40001 - 2/(40001*exp(pi))
```

We can obtain a numerical value for this using the `vpa` function as follows:

```
>> vpa(res)

ans =
0.000047838108134108034810408852920091
```

This result confirms the numerical solution given in Section 4.11.

The following examples require infinite limits. Limits of this type are easily accommodated using the symbols `inf` and `-inf`. Consider the following integrals:

$$y = \int_0^\infty \log_e\left(1 + e^{-x}\right) dx \quad \text{and} \quad y = \int_{-\infty}^\infty \frac{dx}{\left(1 + x^2\right)^2}$$

These can be evaluated as follows:

```
>> syms x, int(log(1+exp(-x)),0,inf)

ans =
pi^2/12

>> syms x, f = 1/(1+x^2)^2;
>> int(f,-inf,inf)

ans =
pi/2
```

These results confirm those given by the numerical integration in Section 4.8.

We now consider what happens when an integral cannot be evaluated by symbolic means. In this case we must resort to numerical procedures to find an approximate numerical value for our integral.

```
>> p = sin(x^3);
>> int(p)
Warning: Explicit integral could not be found.

ans =
int(sin(x^3), x)
```

Note that the integral cannot be evaluated symbolically. If we now insert upper and lower limits for this integral, we have

```
>> int(p,0,1)

ans =
hypergeom([2/3], [3/2, 5/3], -1/4)/4
```

This is the hypergeometric function; see Abramowitz and Stegun (1965) or Olver et al. (2010). It can be evaluated under certain conditions; see MATLAB `help hypergeom`.

Clearly the result is not reduced to a numerical value, but in this case we can use a numerical method to solve it as follows:

```
>> fv = @(x) sin(x.^3);
>> quad(fv,0,1)

ans =
    0.2338
```

We now consider two interesting examples that raise new issues in symbolic processing. Consider the two integrals

$$\int_0^\infty e^{-x}\log_e(x)\,dx \quad \text{and} \quad \int_0^\infty \frac{\sin^4(mx)}{x^2}dx$$

We may evaluate the first integral as follows:

```
>> syms x; int(exp(-x)*log(x),0,inf)

ans =
-eulergamma

y = vpa('-eulergamma',10)

y =
-0.5772156649
```

In MATLAB, `eulergamma` is Euler's constant and is defined by

$$C = \lim_{p\to\infty}\left[-\log_e p + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{p}\right] = 0.577215\ldots$$

Thus MATLAB shows how Euler's constant arises and allows us to evaluate it to any specified number of significant figures.

Now, considering the second integral,

```
>> syms m, int(sin(m*x)^4/x^2,0,inf)
Warning: Explicit integral could not be found.

ans =
piecewise([0 < m, (pi*m)/4], [m in R_, (pi*abs(m))/4],
              [not m in R_, int(sin(x*m)^4/x^2, x = 0..Inf)])
```

This complicated MATLAB result attempts to provide, where possible, an evaluation of the integral for various ranges of m. Finally, we integrate the Dirac delta function from $-\infty$ to ∞:

```
>> int(dirac(x),-inf, inf)

ans =
1
```

This result accords with the definition of the Dirac delta function. We now consider the integral of the Heaviside function from -5 to 3:

```
>> int(heaviside(x),-5,3)

ans =
3
```

This result is as expected.

Symbolic integration can be performed for two variable functions by repeated application of the `int` function. Consider the double integral defined by (4.48) and repeated here for convenience:

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

This can be evaluated symbolically as follows:

```
>> syms x y; f = x^2*y;
>> int(int(f,y,x^2,x^4),x,1,2)

ans =
6466/77
```

which confirms the numerical result given in Section 4.14.2.

9.11 Symbolic Solution of Ordinary Differential Equations

The Symbolic Toolbox can be used to solve, symbolically, first-order differential equations, systems of first-order differential equations, or higher-order differential equations, together with any initial conditions provided. This symbolic solution of differential equations is implemented in MATLAB using the function `dsolve`, and its use is illustrated by a range of examples.

It is important to note that this approach only provides a symbolic solution if one exists. If no solution exists, then the user should apply one of the numerical techniques provided in MATLAB, such as `ode45`.

The general form of a call of the function `dsolve` to solve a differential equation system is

```
sol = dsolve('de1, de2, de3, ... , den, in1, in2, in3, ... , inn');
```

The independent variable is assumed to be `t` unless given by an optional final parameter of `dsolve`. The parameters `de1`, `de2`, `de3` up to `den` stand for the individual differential equations. These must be written in symbolic form using symbolic variables, standard MATLAB operators, and the symbols D, D2, D3, and so on, which represent the first-, second-, third-, and higher-order differential operators, respectively. The parameters in1, in2, in3, in4, and so on, represent the initial conditions for the differential equations, if these conditions are required. An example of how these initial conditions should be written, assuming a dependent variable y, is

```
y(0) = 1,  Dy(0) = 0,  D2y(0) = 9.1
```

which means that the value of y is 1, $dy/dt = 0$, and $d^2y/dt^2 = 9.1$ when $t = 0$. It is important to note that dsolve accepts up to a maximum of 12 input parameters. If initial conditions are required, this is a significant restriction!

The solution is returned to sol as a MATLAB structure, and consequently the names of the dependent variables must be used to indicate the individual components. For example, if g and y are two dependent variables for the differential equation, sol.y gives the solution for the dependent variable y and sol.g gives the solution for the dependent variable g.

To illustrate these points we consider some examples. Consider the following first-order differential equation:

$$\left(1+t^2\right)\frac{dy}{dt}+2ty=\cos t$$

We may solve this without initial conditions using dsolve as follows:

```
>> s = dsolve('(1+t^2)*Dy+2*t*y=cos(t)')

s =
-(C3 - sin(t))/(t^2 + 1)
```

Notice that the solution contains the arbitrary constant C3. If we now solve the same equation using an initial condition, we proceed as follows:

```
>> s = dsolve('(1+t^2)*Dy+2*t*y=cos(t),y(0)=0')

s =
sin(t)/(t^2 + 1)
```

Note that in this case there is no arbitrary constant.

We now solve a second-order system

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

with the initial conditions $y = 0$ and $dy/dx = 1$ at $x = 0$. To solve this differential equation `dsolve` has the form

```
>> dsolve('D2y+y=cos(2*x), Dy(0)=1, y(0)=0','x')

ans =
(2*cos(x))/3 + sin(x) + sin(x)*(sin(3*x)/6 + sin(x)/2) -
  (2*cos(x)*(6*tan(x/2)^2 - 3*tan(x/2)^4 + 1))/(3*(tan(x/2)^2 + 1)^3)

>> simplify(ans)

ans =
sin(x) + (2*sin(x)^2)/3 - (2*sin(x/2)^2)/3
```

Notice that since the independent variable is x, this is indicated in `dsolve` by a final parameter `x` in the list of parameters.

We now solve the fourth-order differential equation

$$\frac{d^4y}{dt^4} = y$$

with initial conditions

$$y = 1,\ dy/dt = 0,\ d^2y/dt^2 = -1,\ d^3y/dt^3 = 0 \quad \text{when} \quad t = \pi/2$$

We again use `dsolve`. In this example `D4` stands for the fourth derivative operator, with respect to t, and so on.

```
>> dsolve('D4y=y, y(pi/2)=1, Dy(pi/2)=0, D2y(pi/2)=-1, D3y(pi/2)=0')

ans =
sin(t)
```

However, we note that if we try to solve the apparently simple problem

$$\frac{dy}{dx} = \frac{e^{-x}}{x}$$

with the initial condition $y = 1$ when $x = 1$, difficulties arise. Applying `dsolve`, we have

```
>> dsolve('Dy=exp(-x)/x, y(1)=1', 'x')

ans =
1 - Ei(1, x) - Ei(-1)
```

Note that `Ei(-1) = -Ei(1,1)`. Clearly this result is not an explicit solution. The function `Ei(1,x)` is the exponential integral and can be found in `mfunlist`. Details of this mathematical function can be found in Abramowitz and Stegun (1965) and Olver et al.

(2010). We can evaluate the function `Ei` using the `mfun` function for any parameters. For example

```
>> y = mfun('Ei',1,1)

y =
    0.2194
```

If we require more digits in the solution, we have

```
vpa('Ei(1,1)',20)

ans =
0.21938393439552027368
```

We will now attempt to solve the following apparently simple differential equation:

$$\frac{dy}{dx} = \cos(\sin x)$$

Applying `dsolve` we have

```
>> dsolve('Dy=cos(sin(x))','x')

ans =
C17 + int(cos(sin(x)), x, IgnoreAnalyticConstraints)
```

In this case `dsolve` fails to solve the equation.

The differential equation may also contain constants represented as symbols. For example, if we wish to solve the equation

$$\frac{d^2x}{dt^2} + \frac{a}{b}\sin t = 0$$

with initial conditions $x = 1$ and $dx/dt = 0$ when $t = 0$, we enter the following:

```
>> syms x t a b
>> x = dsolve('D2x+(a/b)*sin(t)=0,x(0)=1,Dx(0)=0')

x =
(a*sin(t))/b - (a*t)/b + 1
```

Note how the variables a and b appear in the solution as expected.

As an example of solving two simultaneous differential equations, we note that this differential equation may be rewritten as

$$\frac{du}{dt} = -\frac{a}{b}\sin t$$
$$\frac{dx}{dt} = u$$

Using the same initial conditions, `dsolve` may be applied to solve these equations by writing

```
>> syms u
>> [u x] = dsolve('Du+(a/b)*sin(t)=0,Dx=u,x(0)=1,u(0)=0')

u =
(a*cos(t))/b - a/b

x =
(a*sin(t))/b - (a*t)/b + 1
```

This gives the same solution as that obtained from `dsolve` applied directly to solve the second-order differential equation.

The following example provides an interesting comparison of the symbolic and numerical approach. It consists of a script and the output from the script. The script compares the use of `dsolve` for the symbolic solution of a differential equation and the use `ode45` for the numerical solution of the same differential equation. Note that the symbolic solution is obtained in two ways: by solving the second-order equation directly and by separating it into two first-order simultaneous equations. Both approaches provide the same solution.

```
% e3s904.m  Simultaneous first order differential equations
% dx/dt = y, Dy = 3*t-4*x.
% Using dsolve this becomes
syms y t x
x = dsolve('D2x+4*x=3*t','x(0)=0', 'Dx(0)=1')
tt = 0:0.1:5; p = subs(x,t,tt); pp = double(p);
% Plot the symbolic solution to the differential equ'n
plot(tt,pp,'r')
hold on
xlabel('t'), ylabel('x')
sol = dsolve('Dx=y','Dy=3*t-4*x', 'x(0)=0', 'y(0)=1');
sol_x = sol.x, sol_y = sol.y
fv = @(t,x) [x(2); 3*t-4*x(1)];
options = odeset('reltol', 1e-5,'abstol',1e-5);
tspan = [0 5]; initx = [0 1];
[t,x] = ode45(fv,tspan,initx,options);
plot(t,x(:,1),'k+');
axis([0 5 0 4])
```

Executing the script gives the symbolic solution

```
x =
 (3*t)/4 + sin(2*t)/8
```

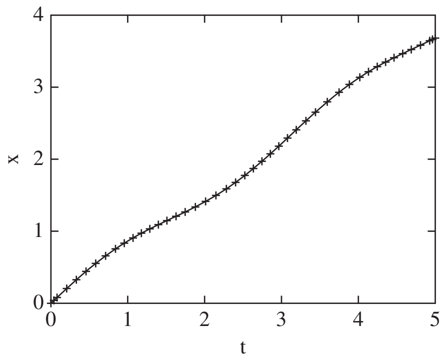

FIGURE 9.3 Symbolic solution and numerical solution indicated by +.

```
sol_x =
(3*t)/4 + sin(2*t)/8

sol_y =
cos(2*t)/4 + 3/4
```

This script also provides the graph of the symbolic solution with alternate numerical solution values also plotted (Figure 9.3). Note how the numerical solution is consistent with the symbolic one.

9.12 The Laplace Transform

The Symbolic Toolbox allows the symbolic determination of the Laplace transforms of many functions. The Laplace transform is used to transform a linear differential equation into an algebraic equation in order to simplify the process of obtaining the solution. It can also be used to transform a system of *differential* equations into a system of *algebraic* equations. The Laplace transform maps a continuous function $f(t)$ in the t domain into the function $F(s)$ in the s domain where $s = \sigma + \jmath\omega$; that is, s is complex. Let $f(t)$ have a finite origin, which can be assumed to be at $t = 0$. In this case we can write

$$F(s) = \int_{0}^{\infty} f(t)e^{-st}dt \tag{9.8}$$

where $f(t)$ is a given function defined for all positive values of t and $F(s)$ is the Laplace transform of $f(t)$. This transform is called the one-sided Laplace transform. The parameter

s is restricted so that the integral converges, and it should be noted that for many functions the Laplace transform does not exist. The inverse transform is given by

$$f(t) = \frac{1}{\jmath 2\pi} \int_{\sigma_0 - \jmath\infty}^{\sigma_0 + \jmath\infty} F(s) e^{st} ds \tag{9.9}$$

where $\jmath = \sqrt{-1}$ and σ_0 is any real value such that the contour $\sigma_0 - \jmath\omega$, for $-\infty < \omega < \infty$, is in the region of convergence of $F(s)$. In practice (9.9) is not used to compute the inverse transform. The Laplace transform of $f(t)$ may be denoted by the operator $\mathcal{L}$. Thus

$$F(s) = \mathcal{L}[f(t)] \quad \text{and} \quad f(t) = \mathcal{L}^{-1}[F(s)]$$

We now give examples of the use of the Symbolic Toolbox for finding Laplace transforms of certain functions.

```
>> syms t
>> laplace(t^4)

ans =
24/s^5
```

We can use a variable other than `t` as the independent variable as follows:

```
>> syms x; laplace(heaviside(x))

ans =
1/s
```

In this brief introduction to the Laplace transform we will not discuss its properties but merely state the following results:

$$\mathcal{L}\left[\frac{df}{dt}\right] = sF(s) - f(0)$$

$$\mathcal{L}\left[\frac{d^2f}{dt^2}\right] = s^2F(s) - sf(0) - f^{(1)}(0)$$

where $f(0)$ and $f^{(1)}(0)$ are the values of $f(t)$ and its first derivative when $t = 0$. This pattern is continued for higher derivatives.

Suppose we wish to solve the following differential equation:

$$\ddot{y} - 3\dot{y} + 2y = 4t + e^{3t}, \quad y(0) = 1, \quad \dot{y}(0) = -1 \tag{9.10}$$

where the dot notation denotes differentiation with respect to time. Taking the Laplace transform of (9.10), we have

$$s^2Y(s) - sy(0) - y^{(1)}(0) - 3\{sY(s) - y(0)\} + 2Y(s) = \mathcal{L}\left[4t + e^{3t}\right] \tag{9.11}$$

Now, from the definition of the Laplace transform or from tables, we can determine $\mathcal{L}[4t + e^{3t}]$. Here we use the Symbolic Toolbox to determine the required transform:

```
>> syms s t
>> laplace(4*t+exp(3*t))

ans =
1/(s - 3) + 4/s^2
```

Substituting this result in (9.11) and rearranging gives

$$(s^2 - 3s + 2)Y(s) = \frac{4}{s^2} + \frac{1}{s-3} - 3y(0) + sy(0) + y^{(1)}(0)$$

Applying the initial conditions and further rearranging, we have

$$Y(s) = \left(\frac{1}{s^2 - 3s + 2}\right)\left(\frac{4}{s^2} + \frac{1}{s-3} - 4 + s\right)$$

To obtain the solution $y(t)$, we must determine the inverse transform of this equation. It has already been stated that in practice (9.9) is not used to compute the inverse transform. The usual procedure is to rearrange the transform into one that can be recognized in tables of Laplace transforms, and typically the method of partial fractions is used for this purpose. However, the MATLAB Symbolic Toolbox allows us to avoid this task and determine inverse transforms using the `ilaplace` statement as follows:

```
>> ilaplace((4/s^2+1/(s-3)-4+s)/(s^2-3*s+2))

ans =
2*t - 2*exp(2*t) + exp(3*t)/2 - exp(t)/2 + 3
```

Thus $y(t) = 2t + 3 + 0.5(e^{3t} - e^t) - 2e^{2t}$.

9.13 The *Z*-Transform

The Z-transform plays a similar role to the Laplace transform in the solution of difference equations representing discrete systems. The Z-transform is defined by

$$F(z) = \sum_{n=0}^{\infty} f_n z^{-n} \tag{9.12}$$

where f_n is a sequence of data beginning at f_0. The function $F(z)$ is called the unilateral or single-sided Z-transform of f_n and is denoted $\mathcal{Z}[f_n]$. Thus

$$F(z) = \mathcal{Z}[f_n]$$

The inverse transform is denoted by $\mathcal{Z}^{-1}[F(z)]$. Thus

$$f_n = \mathcal{Z}^{-1}[F(z)]$$

Like the Laplace transform, the Z-transform has many important properties. These properties will not be discussed here, but we do provide the following important results:

$$\mathcal{Z}\left[f_{n+k}\right] = z^k F(z) - \sum_{m=0}^{k-1} z^{k-m} f_m \tag{9.13}$$

$$\mathcal{Z}\left[f_{n-k}\right] = z^{-k} F(z) + \sum_{m=1}^{k} z^{-(k-m)} f_{(-m)} \tag{9.14}$$

These are the left- and right-shifting properties, respectively.

We can use the Z-transform to solve *difference* equations in much the same way as we use the Laplace transform to solve *differential* equations. For example, consider the following difference equation:

$$6y_n - 5y_{n-1} + y_{n-2} = \frac{1}{4^n}, \quad n \geq 0 \tag{9.15}$$

Here y_n is a sequence of data values beginning at y_0. However, when $n = 0$ in (9.15), we require the values of y_{-1} and y_{-2} to be specified. These are *initial conditions* and play a similar role to the initial conditions in a differential equation. Let the initial conditions be $y_{-1} = 1$ and $y_{-2} = 0$. Taking $k = 1$ in (9.14), we have

$$\mathcal{Z}\left[y_{n-1}\right] = z^{-1} Y(z) + y_{-1}$$

and taking $k = 2$ in (9.14), we have

$$\mathcal{Z}\left[y_{n-2}\right] = z^{-2} Y(z) + z^{-1} y_{-1} + y_{-2}$$

Taking the Z-transform of (9.15), and substituting for the transformed values of y_{-1} and y_{-2}, we have

$$6Y(z) - 5\{z^{-1}Y(z) + y_{-1}\} + \{z^{-2}Y(z) + z^{-1}y_{-1} + y_{-2}\} = \mathcal{Z}\left[\frac{1}{4^n}\right] \tag{9.16}$$

We could use the basic definition of the Z-transform or tabulated relationships to determine the Z-transform of the right side of this equation. However, the MATLAB Symbolic

Toolbox gives the Z-transform of a function as follows:

```
>> syms z n
>> ztrans(1/4^n)

ans =
z/(z - 1/4)
```

Substituting this result in (9.16), we have

$$(6-5z^{-1}+z^{-2})Y(z)=\frac{4z}{4z-1}-z^{-1}y_{-1}-y_{-2}+5y_{-1}$$

Substituting for y_{-1} and y_{-2} gives

$$Y(z)=\left(\frac{1}{6-5z^{-1}+z^{-2}}\right)\left(\frac{4z}{z-1}-z^{-1}+5\right)$$

We can determine y_n by taking the inverse of the Z-transform. Using the MATLAB function `iztrans` we have

```
>> iztrans((4*z/(4*z-1)-z^(-1)+5)/(6-5*z^(-1)+z^(-2)))

ans =
(5*(1/2)^n)/2 - 2*(1/3)^n + (1/4)^n/2
```

Thus

$$y_n=\frac{5}{2}\left(\frac{1}{2}\right)^n-2\left(\frac{1}{3}\right)^n+\frac{1}{2}\left(\frac{1}{4}\right)^n$$

Evaluating this solution for $n=-2$ and -1 shows that it satisfies the initial conditions.

9.14 Fourier Transform Methods

Fourier analysis transforms data or functions from the time or spatial domain into the frequency domain. Here, we will transform from the x domain into the ω domain because MATLAB uses `x` and `w` (corresponding to ω) as the default parameters in the implementation of the Fourier and inverse Fourier transforms.

The Fourier series transforms a periodic function in the x domain into corresponding discrete values in the frequency domain; these are the Fourier coefficients. In contrast, the Fourier transform takes a nonperiodic and continuous function in the x domain and transforms it into a infinite and continuous function in the frequency domain.

The Fourier transform of a function $f(x)$ is given by

$$F(s)=\mathcal{F}[f(x)]=\int_{-\infty}^{\infty} f(x)e^{-sx}dx \tag{9.17}$$

where $s = \jmath\omega$; that is, s is imaginary. Thus we may write

$$F(\omega) = \mathcal{F}[f(x)] = \int_{-\infty}^{\infty} f(x)e^{-\jmath\omega x}dx \tag{9.18}$$

$F(\omega)$ is complex and is called the frequency spectrum of $f(x)$. The inverse Fourier transform is given by

$$f(x) = \mathcal{F}^{-1}[F(\omega)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{\jmath\omega x}d\omega \tag{9.19}$$

Here we use the operators $\mathcal{F}$ and $\mathcal{F}^{-1}$ to indicate the Fourier transform and the inverse Fourier transform, respectively. Not all functions have a Fourier transform, and certain conditions must be met if the Fourier transform is to exist (see Bracewell, 1978). The Fourier transform has important properties that are introduced as appropriate.

We will begin by using the MATLAB symbolic function `fourier` to determine the Fourier transform of $\cos(3x)$.

```
>> syms x, y = fourier(cos(3*x))

y =
pi*(dirac(w - 3) + dirac(w + 3))
```

This Fourier transform pair is shown diagrammatically in Figure 9.4. The Fourier transform tells us that the frequency spectrum of this cosine function consists of two infinitely narrow components, one at $\omega = 3$ and one at $\omega = -3$ (for a description of the Dirac delta function, see Section 9.7). The MATLAB function `ifourier` implements the symbolic inverse Fourier transform as follows:

```
>> z = ifourier(y)

z =
1/(2*exp(x*3*i)) + exp(x*3*i)/2

>> simplify(z)

ans =
cos(3*x)
```

As a second example of the use of the Fourier transform, consider the transform of the function shown in Figure 9.5, which has a unit value in the range $-2 < x < 2$ and zero elsewhere. Note how this has been constructed from two Heaviside functions (the Heaviside function is described in Section 9.7).

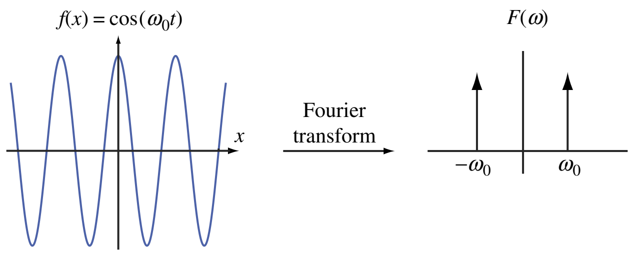

FIGURE 9.4 The Fourier transform of a cosine function.

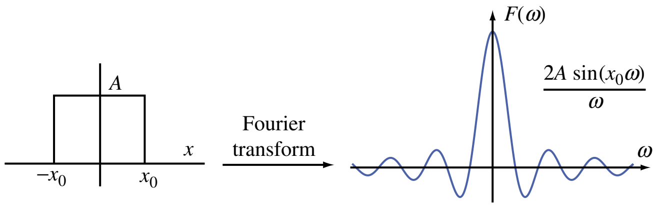

FIGURE 9.5 The Fourier transform of a "top-hat" function.

```
>> syms x
>> fourier(heaviside(x+2) - heaviside(x-2))

ans =
(cos(2*w)*i + sin(2*w))/w - (cos(2*w)*i - sin(2*w))/w
```

This expression can be simplified as follows:

```
simplify(ans)

ans =
(2*sin(2*w))/w
```

Note that the original function, which in the x domain is limited to the range $-2 < x < 2$, has a frequency spectrum that is continuous between $-\infty < \omega < \infty$. This is shown in Figure 9.5.

We now illustrate the use of the Fourier transform in the solution of a partial differential equation. Consider the equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \quad (-\infty < x < \infty,\ t > 0) \tag{9.20}$$

subject to the initial condition

$$u(x, 0) = \exp(-a^2 x^2) \quad \text{where} \quad a = 0.1$$

It can be proved that

$$\mathcal{F}\left[\frac{\partial^2 u}{\partial x^2}\right] = -\omega^2 \mathcal{F}[u] \quad \text{and} \quad \mathcal{F}\left[\frac{\partial u}{\partial t}\right] = \frac{\partial}{\partial t}\{\mathcal{F}[u]\}$$

Thus, taking the Fourier transform of (9.20), we have

$$\frac{\partial}{\partial t}\{\mathcal{F}[u]\} + \omega^2 \mathcal{F}[u] = 0$$

Solving this first-order differential equation for $\mathcal{F}[u]$ gives

$$\mathcal{F}[u] = A\exp(-\omega^2 t) \tag{9.21}$$

To determine the constant A we must apply the initial conditions. We begin by using MATLAB to find the Fourier transform of the initial conditions:

```
>> syms x y z w
>> z = fourier(exp(-x^2/100))

z =
(10*pi^(1/2))/exp(25*w^2)
```

Hence

$$\mathcal{F}[u(x,0)] = \sqrt{100\pi}\exp(-25\omega^2)$$

Comparing this equation with (9.21) when $t = 0$, we see that

$$A = \sqrt{100\pi}\exp(-25\omega^2)$$

Substituting this result in (9.21), we have

$$\mathcal{F}[u] = \sqrt{100\pi}\exp(-25\omega^2)\exp(-\omega^2 t)$$

Taking the inverse transform of this equation gives

$$u(x,t) = \mathcal{F}^{-1}\left[\sqrt{100\pi}\exp(-25\omega^2)\exp(-\omega^2 t)\right]$$

Suppose we require a solution when $t = 4$. Using MATLAB to compute the inverse Fourier transform, we have

```
>> y = z*exp(-4*w^2)

y =
(10*pi^(1/2))/exp(29*w^2)
```

```
>> ifourier(y)

ans =
(5*29^(1/2))/(29*exp(x^2/116))
```

Thus, when $t = 4$, the solution of (9.20) is

$$u(x,4) = \frac{5\sqrt{29}}{29}\exp(-x^2/116)$$

Consider now the Fourier transform of the Heaviside or step function.

```
>> syms x
>> fourier(heaviside(x))

ans =
pi*dirac(w) - i/w
```

Thus the real part of the Fourier transform of the Heaviside function is π times the Dirac delta function at $\omega = 0$ and the imaginary part is $1/\omega$, which tends to plus and minus infinity when $\omega = 0$. The step function has many applications. For example, if we require the Fourier transform of the function

$$f(x) = \begin{cases} e^{-2x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

using the Heaviside or unit step function, we can rewrite this as

$$f(x) = u(x)e^{-2x} \quad \text{for all} \quad x$$

where $u(x)$ is the Heaviside function. The MATLAB implementation is

```
>> syms x
>> fourier(heaviside(x)*exp(-2*x))

ans =
1/(w*i + 2)
```

9.15 Linking Symbolic and Numerical Processes

Symbolic algebra can be used to ease the burden for the user in the numerical solution process. To illustrate this we show how the Symbolic Toolbox can be used in a version of Newton's method for solving a nonlinear equation that only requires the user to supply the function itself. The usual implementation of this algorithm (see Section 3.7) requires the user to supply the first-order derivative of the function as well as the function itself. The modified algorithm takes the following form in MATLAB:

```
function [res, it] = fnewtsym(func,x0,tol)
% Finds a root of f(x) = 0 using Newton's method
% using the symbolic toolbox.
% Example call: [res, it] = fnewtsym(func,x,tol)
% The user defined function func is the function f(x) which must
% be defined as a symbolic function.
% x is an initial starting value, tol is required accuracy.
it = 1; syms dfunc x
% Now perform the symbolic differentiation:
dfunc = diff(sym(func));
d = double(subs(func,x,x0)/subs(dfunc,x,x0));
while abs(d)>tol
    x1 = x0-d; x0 = x1;
    d = double(subs(func,x,x0)/subs(dfunc,x,x0));
    it = it+1;
end
res = x0;
```

Notice the use of the `subs` and `double` functions so that a numerical value is returned. To illustrate the use of `fnewtsym` we will solve $\cos x - x^3 = 0$ to find the root closest to 1 with an accuracy to four decimal places.

```
>> [r,iter] = fnewtsym('cos(x)-x^3',1,0.00005)

r =
    0.8655

iter =
     4
```

These results are identical to those obtained using function `fnewton` (see Section 3.7), but here the user is not required to provide the derivative of the function.

The following examples provide further illustrations of how the Symbolic Toolbox can help users perform the routine, tedious, and sometimes difficult tasks required by some numerical methods. We have seen how the single-variable Newton's method can be modified using symbolic differentiation; now we extend this to Newton's multivariable method to solve a system of equations. Here the use of symbolic functions provides an even greater savings for users. The equations solved in this example are

$$x_1 x_2 = 2$$
$$x_1^2 + x_2^2 = 4$$

The MATLAB function takes the form

```
function [x1,fr,it] = newtmvsym(x,f,n,tol)
% Newton's method for solving a system of n nonlinear equations
% in n variables. This version is restricted to two variables.
% Example call: [xv,it] = newtmvsym(x,f,n,tol)
% Requires an initial approximation column vector x. tol is
% required accuracy.
% User must define functions f, the system equations.
% xv is the solution vector, parameter it is number of iterations.
syms a b
xv = sym([a b]); it = 0;
fr = double(subs(f,xv,x));
while norm(fr)>tol
    Jr = double(subs(jacobian(f,xv),xv,x));
    x1 = x-(Jr\fr')'; x = x1;
    fr = double(subs(f,xv,x1));
    it = it+1;
end
```

Notice how this function uses the symbolic Jacobian function in the line

```
Jr = double(subs(jacobian(f,xv),xv,x));
```

This ensures that the user is not required to find the four partial derivatives required in this case. To use this function we run the script

```
% e3s905.m.  Script for running newtonmvsym.m
syms a b
x = sym([a b]);
format long
f = [x(1)*x(2)-2,x(1)^2+x(2)^2-4];
[x1,fr,it] = newtmvsym([1 0],f,2,.000000005)
```

Running this script gives

```
x1 =
   1.414244079950892   1.414183044795298

fr =
  1.0e-008 *
  -0.093132257461548   0.186264514923096

it =
    14
```

We note that this result provides an accurate solution for the given problem in two variables.

It is interesting to note that in a similar way we can write a script for the conjugate gradient method that enables the user to avoid having to provide the first-order partial derivatives of the function to be minimized. This script uses the statements

```
for i = 1:n, dfsymb(i) = diff(sym(f),xv(i)); end
df = double(subs(dfsymb,xv(1:n),x(1:n)'));
```

to obtain the gradient of the function where required. To run this modified function a script similar to that given earlier for the multivariable Newton's method must be written. This script defines the nonlinear function to be optimized and defines a symbolic vector `x`.

9.16 Summary

We have introduced a wide range of symbolic functions and shown how they can be applied to such standard mathematical processes as integration, differentiation, expansion, and simplification. We have also shown how symbolic methods can sometimes be directly linked to numerical procedures with good effect. For those with access to the Symbolic Toolbox, care must be taken to choose the appropriate method, symbolic or numerical, for the problem.

Problems

9.1. Using the appropriate MATLAB symbolic function, rearrange the following expression as a polynomial in x:

$$\left(x-\frac{1}{a}-\frac{1}{b}\right)\left(x-\frac{1}{b}-\frac{1}{c}\right)\left(x-\frac{1}{c}-\frac{1}{a}\right)$$

9.2. Using the appropriate MATLAB symbolic function, multiply the polynomials

$$f(x)=x^4+4x^3-17x^2+27x-19 \quad \text{and} \quad g(x)=x^2+12x-13$$

and simplify the resulting expression. Then arrange the solution in a nested form.

9.3. Using the appropriate MATLAB symbolic function, expand the following functions:

(a) $\tan(4x)$ in terms of powers of $\tan(x)$
(b) $\cos(x+y)$ in terms of $\cos(x)$, $\cos(y)$, $\sin(x)$, $\sin(y)$
(c) $\cos(3x)$ in terms of powers of $\cos(x)$
(d) $\cos(6x)$ in terms of powers of $\cos(x)$.

9.4. Using the appropriate MATLAB symbolic function, expand $\cos(x+y+z)$ in terms of $\cos(x)$, $\cos(y)$, $\cos(z)$, $\sin(x)$, $\sin(y)$, and $\sin(z)$.

9.5. Using the appropriate MATLAB symbolic function, expand the following functions in ascending powers of x up to x^7:

(a) $\sin^{-1}(x)$
(b) $\cos^{-1}(x)$
(c) $\tan^{-1}(x)$

9.6. Using the appropriate MATLAB symbolic function, expand $y = \log_e(\cos(x))$ in ascending powers of x up to x^{12}.

9.7. The first three terms of a series are

$$\frac{4}{1\cdot 2\cdot 3}\left(\frac{1}{3}\right)+\frac{5}{2\cdot 3\cdot 4}\left(\frac{1}{9}\right)+\frac{6}{3\cdot 4\cdot 5}\left(\frac{1}{27}\right)+\cdots$$

Sum this series to n terms using the MATLAB functions `symsum` and `simple`.

9.8. Using the appropriate MATLAB symbolic function, sum the first hundred terms of the series whose kth term is k^{10}.

9.9. Using the appropriate MATLAB symbolic function, verify that

$$\sum_{k=1}^{\infty} k^{-4} = \frac{\pi^4}{90}$$

9.10. For the matrix

$$\mathbf{A} = \begin{bmatrix} 1 & a & a^2 \\ 1 & b & b^2 \\ 1 & c & c^2 \end{bmatrix}$$

determine $\mathbf{A}^{-1}$ symbolically using the MATLAB function `inv` and, using the function `factor`, express it in the following form:

$$\begin{bmatrix} \dfrac{cb}{(a-c)(a-b)} & \dfrac{-ac}{(b-c)(a-b)} & \dfrac{ab}{(b-c)(a-c)} \\ \dfrac{-(b+c)}{(a-c)(a-b)} & \dfrac{(a+c)}{(b-c)(a-b)} & \dfrac{-(a+b)}{(b-c)(a-c)} \\ \dfrac{1}{(a-c)(a-b)} & \dfrac{-1}{(b-c)(a-b)} & \dfrac{1}{(b-c)(a-c)} \end{bmatrix}$$

9.11. Using the appropriate MATLAB symbolic function, show that the characteristic equation of the matrix

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

is

$$\lambda^4 - a_1\lambda^3 - a_2\lambda^2 - a_3\lambda - a_4 = 0$$

9.12. The rotation matrix **R** follows. Define it symbolically and find its second and fourth powers using MATLAB.

$$\mathbf{R} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

9.13. Solve, symbolically, the general cubic equation having the form $x^3 + 3hx + g = 0$. *Hint*: Use the MATLAB function `subexpr` to simplify your result.

9.14. Using the appropriate MATLAB symbolic function, solve the cubic equation $x^3 - 9x + 28 = 0$.

9.15. Using the appropriate MATLAB symbolic functions, find the roots of $z^6 = 4\sqrt{2}(1 + \jmath)$ and plot these roots in the complex plane. *Hint*: Convert the answer using `double`.

9.16. Using the appropriate MATLAB symbolic function, differentiate the following function with respect to x:

$$y = \log_e \left\{ \frac{(1-x)(1+x^3)}{1+x^2} \right\}$$

9.17. Given that Laplace's equation is

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

use the appropriate MATLAB symbolic function to verify that this equation is satisfied by the following functions:

(a) $z = \log_e(x^2 + y^2)$
(b) $z = e^{-2y}\cos(2x)$

9.18. Using the appropriate MATLAB symbolic function, verify that $z = x^3 \sin y$ satisfies the following conditions:

$$\frac{\partial^2 z}{\partial x \partial y} = \frac{\partial^2 z}{\partial y \partial x} \quad \text{and} \quad \frac{\partial^{10} z}{\partial x^4 \partial y^6} = \frac{\partial^{10} z}{\partial y^6 \partial x^4} = 0$$

9.19. Using the appropriate MATLAB symbolic functions, determine the integrals of the following functions and then differentiate the result to recover the original function:

$$\textbf{(a)} \quad \frac{1}{(a+fx)(c+gx)} \qquad \textbf{(b)} \quad \frac{1-x^2}{1+x^2}$$

9.20. Using the appropriate MATLAB symbolic function, determine the following indefinite integrals:

$$\textbf{(a)} \quad \int \frac{1}{1+\cos x+\sin x}dx \qquad \textbf{(b)} \quad \int \frac{1}{a^4+x^4}dx$$

9.21. Using the appropriate MATLAB symbolic function, verify the following result:

$$\int_0^\infty \frac{x^3}{e^x-1}dx = \frac{\pi^4}{15}$$

9.22. Using the appropriate MATLAB symbolic function, evaluate the following integrals:

$$\textbf{(a)} \quad \int_0^\infty \frac{1}{1+x^6}dx \qquad \textbf{(b)} \quad \int_0^\infty \frac{1}{1+x^{10}}dx$$

9.23. Using the appropriate MATLAB symbolic function, evaluate the integral

$$\int_0^1 \exp(-x^2)dx$$

by expanding $\exp(-x^2)$ in an ascending series of powers of x and then integrating term by term to obtain a series approximation. Expand the series to both x^6 and x^{14} and hence find two approximations to the integral. Compare the accuracy of your results. The solution to 10 decimal places is 0.7468241328.

9.24. Using the appropriate MATLAB symbolic function, verify the following result:

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

9.25. Using the appropriate MATLAB symbolic function, evaluate the integral

$$\int_0^1 \log_e(1+\cos x)dx$$

by expanding $\log_e(1+\cos x)$ in an ascending series of powers of x and then integrating term by term to obtain a series approximation. Expand the series up to the term in x^4. Compare the accuracy of your result. The solution to 10 decimal places is 0.6076250333.

9.26. Using the appropriate MATLAB symbolic function, evaluate the following repeated integral:

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

9.27. Using the appropriate MATLAB symbolic function, solve the following differential equation, which arises in the study of consumer behavior:

$$\frac{d^2y}{dt^2} + (bp + aq)\frac{dy}{dt} + ab(pq-1)y = cA$$

Also find the solution in the case where $p = 1$, $q = 2$, $a = 2$, $b = 1$, $c = 1$, and $A = 20$. *Hint*: Use the MATLAB function `subs`.

9.28. Using the appropriate MATLAB symbolic function, solve the following pair of simultaneous differential equations:

$$2\frac{dx}{dt} + 4\frac{dy}{dt} = \cos t$$
$$4\frac{dx}{dt} - 3\frac{dy}{dt} = \sin t$$

9.29. Using the appropriate MATLAB symbolic function, solve the following differential equation:

$$(1-x^2)\frac{d^2y}{dx^2} - 2x\frac{dy}{dx} + 2y = 0$$

9.30. Using the appropriate MATLAB symbolic function, solve the following differential equations using the Laplace transform:

(a) $\frac{d^2y}{dt^2} + 2y = \cos(2t), \quad y = -2 \quad \text{and} \quad \frac{dy}{dt} = 0 \quad \text{when} \quad t = 0$

(b) $\frac{dy}{dt} - 2y = t, \quad y = 0 \quad \text{when} \quad t = 0$

(c) $\frac{d^2y}{dt^2} - 3\frac{dy}{dt} + y = \exp(-2t), \; y = -3 \quad \text{and} \quad \frac{dy}{dt} = 0 \quad \text{when} \quad t = 0$

(d) $\frac{dq}{dt} + \frac{q}{c} = 0, \quad q = V \quad \text{when} \quad t = 0$

9.31. Solve the following difference equations symbolically using the Z-transform:

(a) $y_n + 2y_{n-1} = 0, \quad y_{-1} = 4$

(b) $y_n + y_{n-1} = n, \quad y_{-1} = 10$

(c) $y_n - 2y_{n-1} = 3, \quad y_{-1} = 1$

(d) $y_n - 3y_{n-1} + 2y_{n-2} = 3(4^n), \quad y_{-1} = -3, \quad y_{-2} = 5$