Appendix B
MATLAB Tutorial

This tutorial is an introduction to MATLAB. MATLAB is an interactive environment for scientific and engineering calculations, design, simulation and visualization. The aim of this tutorial is to enable students and control engineers to learn to use MATLAB in control engineering applications. The tutorial provides a brief introduction to the use of MATLAB with examples, and then describes the *Control System Toolbox* with examples. With the aid of this toolbox, for example, the control engineer can draw the Bode diagram, root locus or time response of a system in a few seconds and analyse and design a control system in a very short time.

B.1 MATLAB OPERATIONS

A variable in MATLAB can be a scalar, a complex number, a vector, or a matrix. A variable name can be up to 31 characters long and must start with a letter. It can contain letters, numbers, and underscore characters.

If a data entry, a statement, or any command is not terminated by a semicolon, the result of the statement is always displayed.

An integer number can be assigned to a variable name as follows:

```
>> w = 5;
>> p = -3;
```

A real number is entered by specifying a decimal point, or the exponent:

```
>> q = 2.35;
>> a = 5.2e-3;
```

When entering a complex number, the real part is entered first, followed by the imaginary part:

```
>> p = 2 + 4*i;
>> q = 12*exp (i*2);
```

Microcontroller Based Applied Digital Control D. Ibrahim
 ISBN: 0-470-86335-8

A row vector is entered by optionally separating the elements with commas. For example, the vector

$$p = [2 \quad 4 \quad 6]$$

is entered as

```
>> p = [2, 4, 6];
```

or

```
>> p = [2 4 6];
```

Similarly, a column vector is entered by separating the elements with semicolons. For example, the vector

$$q = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix}$$

is entered as

```
>> q = [3; 6; 9];
```

A vector can be transposed by using the character ‘’’. For example, the vector q above can be transposed as

```
>> a = [3; 6; 9]';
```

to give

```
a = [3  6  9].
```

Matrices are entered similarly to vectors: the elements of each row can be entered by separating them with commas; the rows are then separated by the semicolon character. For example, the 3×3 matrix A given by

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 0 & 6 \\ 5 & 8 & 7 \end{bmatrix}$$

is entered as

```
>> A = [2, 1, 3; 4, 0, 6; 5, 8, 7];
```

or

```
>> A = [2 1 3; 4 0 6; 5 8 7];
```

Special vectors and matrices. MATLAB allows special vectors and matrices to be declared. Some examples are given below:

```
A = []          generates a null matrix
A = ones(n,m)   generates an n × m matrix of ones

A = eye(n)      generates an n × n identity matrix
A = zeros(n,m)  generates an n × m matrix of zeros
```

Some examples are given below:

```
>> A = ones(3,5)
```

gives

```
A =
     1  1  1  1  1
     1  1  1  1  1
     1  1  1  1  1
```

```
>> B = zeros(2,3)
```

gives

```
B =
     0  0  0
     0  0  0
```

and

```
>> C = eye(3,3)
```

gives

```
C =
     1  0  0
     0  1  0
     0  0  1
```

A particular element of a matrix can be assigned by specifying the row and the column numbers:

```
>> A(2,3) = 8;
```

places the number 8 in the second row, third column.

Matrix elements can be accessed by specifying the row and the column numbers:

```
>> C = A(2,1);
```

assigns the value in the second row, first column of A to C.

Vectors can be created by specifying the initial value, final value and increment. For example,

```
>> T = 0:1:10;
```

creates a row vector called T with the elements:

```
T = [0  1  2  3  4  5  6  7  8  9  10].
```

If the increment is 1 it can be omitted. Thus the above vector can also be created with the statement

```
>> T = 0:10;
```

The size of a matrix can be found by using the *size* statement:

```
>> [m,n] = size(C)

m =
     4
n =
     3
```

Arithmetic operators. MATLAB utilizes the following arithmetic operators:

```
+          addition
-          subtraction
*          multiplication
/          division
^          power operator
'          transpose
```

If x is a vector, its multiplication with a scalar multiplies all elements of the vector. For example,

```
>> x = [1,  3,  5];
>> y = 2*x
y =
    2  6  10
```

Similarly, if A is a matrix, its multiplication with a scalar multiplies all elements of the matrix:

```
>> A = [1  3;  5  7];
>> B = 2*A

B =
    2   6
    10  14
```

Two matrices can be multiplied to produce another matrix. For example, if

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 2 & 4 \\ 5 & 2 \end{bmatrix}$$

then

```
>> A = [1  3;  2  4];
>> B = [2  4;  5  2];
>> C = A*B

C =
    17  10
    24  16
```

Array operations perform arithmetic operations in an element-by-element manner. An array operation is indicated by proceeding the operator by a period (.). For example, if $a = [1 \quad 3 \quad 4]$ and b = [2 3 5] then

```
>> a = [1  3  4];
>> b = [2  3  5];
>> c = a.*b

c =
   2  9  20
```

Predefined functions. There are a number of predefined functions that can be used in statements. Some commonly used functions are:

abs	absolute value
sqrt	square root
real	real part
imag	imaginary part
rem	remainder
sin	sine
cos	cosine
asin	arcsine
acos	arccosine
tan	tangent
atan	arctangent
exp	exponential base e
log	natural logarithm
log10	log base 10

For example,

```
>> a = sqrt(16)
a = 4
>> a = sqrt(-4)
a = 0 + 2.0000i
```

Polynomials. A polynomial is defined by using a vector containing the coefficients of the polynomial. For example, the polynomial

$$F(x) = 3x^4 - 5x^3 + x^2 - 3x + 1$$

is defined as

```
p = [3  -5  1  -3  1].
```

It is important that all powers of the polynomial must be specified. The coefficients of the missing powers must be specified as zero.

The following operations can be performed on a polynomial:

roots(p)	find the roots of the polynomial
polyval(p,x)	evaluate the polynomial p at the value of x
deconv(p1,p2)	compute the quotient of p_1 divided by p_2
conv(p1,p2)	compute the product of polynomials p_1 and p_2
poly(r)	compute the polynomial from the vector of $r$oots
poly2str(p,'s')	display the polynomial as an equation in s

For example, consider the polynomial P_1, where,

$$P_1 = 6x^4 - 2x^3 + 5x^2 - 2x + 1.$$

The roots of $P_1 = 0$ are found as follows:

```
>> P1 = [6  -2  5  -2  1]'
>> r = roots(P1)

r =
    -0.1026 + 0.8355i
    -0.1026 - 0.8355i
    0.2692 + 0.4034i
    0.2692 - 0.4034i
```

The polynomial has four complex roots.

The value of the polynomial at $x = 1.2$ is 14.7856 and can be found as follows:

```
>> polyval(P1, 1.2)

ans =

     14.7856
```

The polynomial P_1 can be expressed as an equation in s as:

```
>> poly2str(P1,'s')

ans =
     6 s^4 - 2 s^3 + 5s^2 - 2 s + 1
```

Now consider another polynomial

$$P_2 = 2x^4 - x^3 + 2x^2 - x + 3.$$

The product of the polynomials P1 and P2 can be found as follows:

```
>> P2 = [2  -1  2  -1  3];
>> P3 = conv(P1,P2)

P3 =
     12  -10  24  -19  34  -16  19  -7  3

or

>> P3 = poly2str(conv(P1,P2),'x')

P3 =
   12 x^8 - 10 x^7 + 24 x^6 - 19 x^5 + 34 x^4 - 16 x^3 + 19 x^2 -
      7 x + 3
```

Finally, the polynomial whose roots are 2 and 5 can be found as follows:

```
>> poly([2 5])

ans =

      1  -7  10
```

```
or

>> poly2str(poly([2  5]),'x')

ans =

     x^2 - 7x + 10
```

Thus, the equation of the required polynomial is

$$F(x) = x^2 - 7x + 10.$$

B.2 CONTROL SYSTEM TOOLBOX

The Control System Toolbox is a collection of algorithms and uses MATLAB functions to provide specilized functions in control engineering. In this section we will briefly look at some of the important functions of the Control System Toolbox for both continuous-time and discrete-time systems.

B.2.1 Continuous-Time Systems

Consider a single-input, single-output continuous-time system with the open-loop transfer function

$$\frac{Y(s)}{U(s)} = \frac{3}{s^2 + 3s + 9}.$$

Transfer function. The transfer function of the system can be defined in terms of the numerator and the denominator polynomial:

```
>> num = [0  0  3];
>> den = [1  3  9];
```

The transfer function is given by:

```
>> G = tf(num,den)
```

Transfer function:

```
     3
------------
s^2 + 3 s + 9
```

Step response. The step response is given by

```
>> step(num,den)
```

which produces the plot shown in Figure B.1.

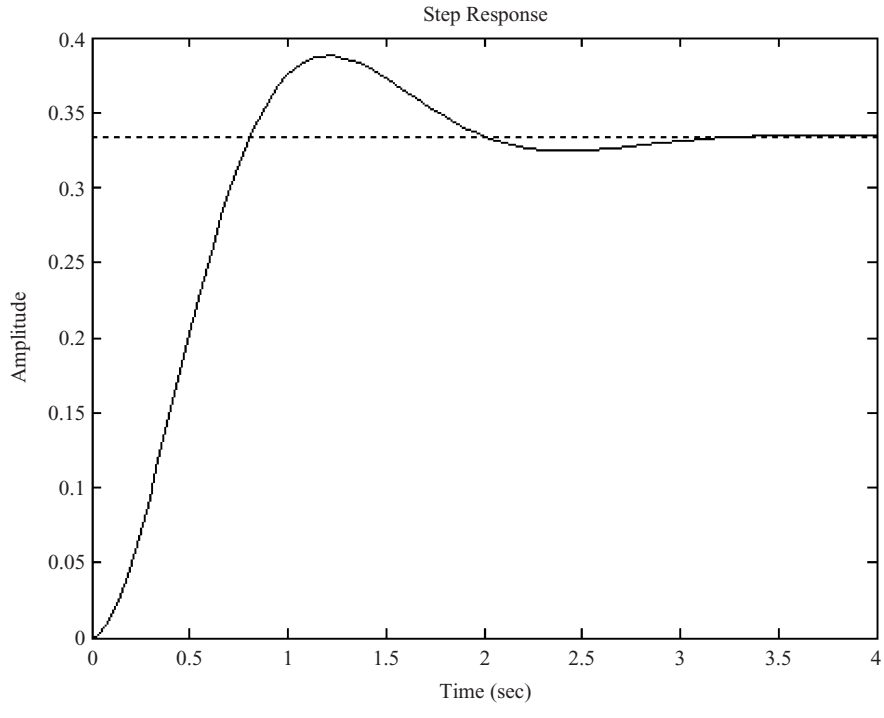

Figure B.1 Step response

The steady-state value of the step response is obtained as follows:

```
>> ys = decgain(num,den)

ys =

    0.3333
```

Impulse response. The impulse response is given by

```
>> impulse(num,den)
```

which produces the plot shown in Figure B.2.

Bode diagram. The Bode diagram can be obtained by writing:

```
>> impulse(num,den);
>> bode(num,den);
>> grid
```

Notice that the *grid* command produces a grid in the display. The Bode diagram of the system is shown in Figure B.3.

Nyquist diagram. The Nyquist diagram can be obtained from

```
>> nyquist(num,den)
```

The Nyquist diagram of the system is shown in Figure B.4.

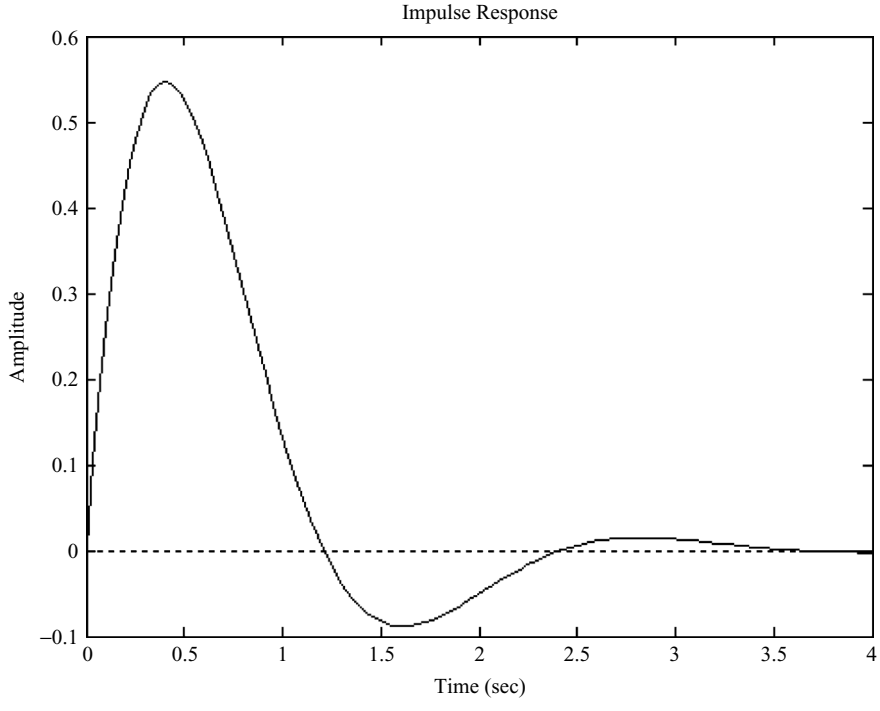

Figure B.2 Impulse response

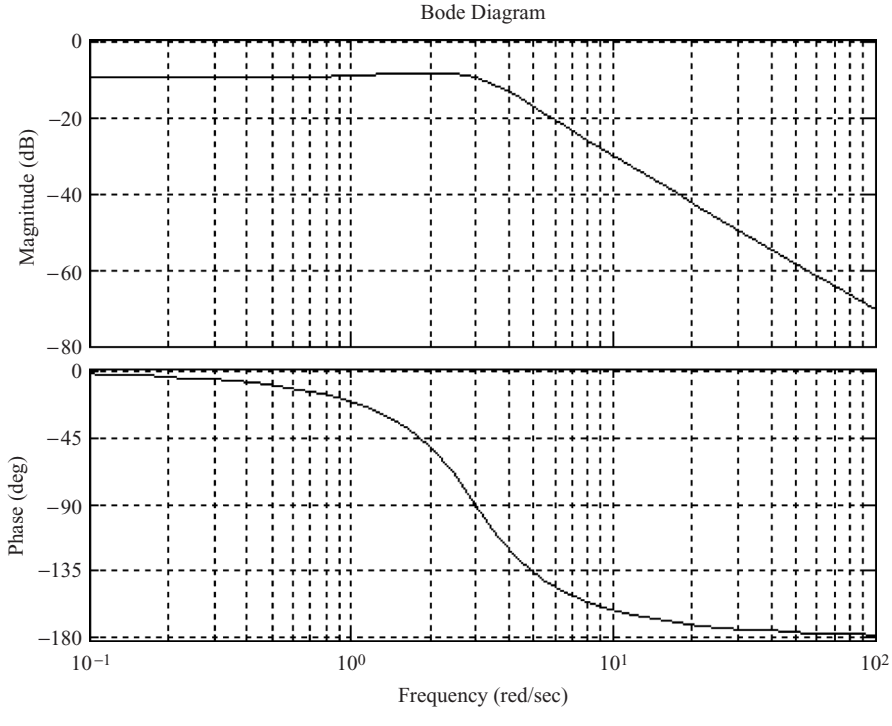

Figure B.3 Bode diagram

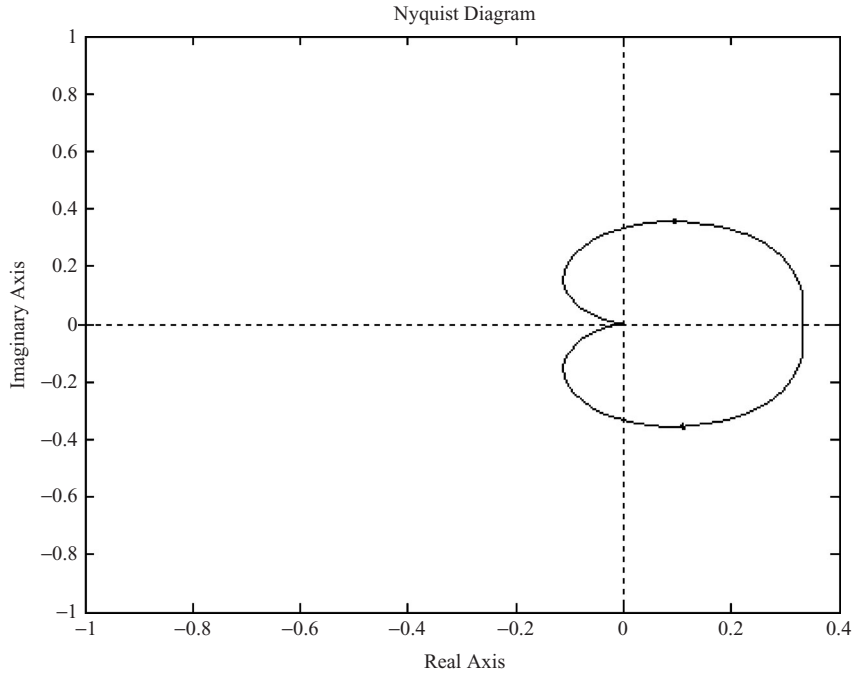

Figure B.4 Nyquist diagram

Nichols diagram. The Nichols diagram can be obtained by writing

```
>> nichols(num,den);
>> grid
```

Figure B.5 shows the Nichols diagram obtained.

Root locus. The root locus diagram of the system is given by

```
>> rlocus(num,den)
```

Figure B.6 shows the graph obtained from this command.

To customize the plot for a specific range of K, say for K ranging from 0 to 5 in steps of 0.5, we can write

```
>> K = 0:0.5:5;
>> r = rlocus(num,den,K);
>> plot(r,'x')
```

Figure B.7 shows the graph obtained, where the character x is plotted at the roots of the system as K is varied.

Zeros and poles. The zero and pole locations can be found from

```
>> [z,p,k] = tf2zp(num,den)

z =
    Empty matrix: 0-by-1
```

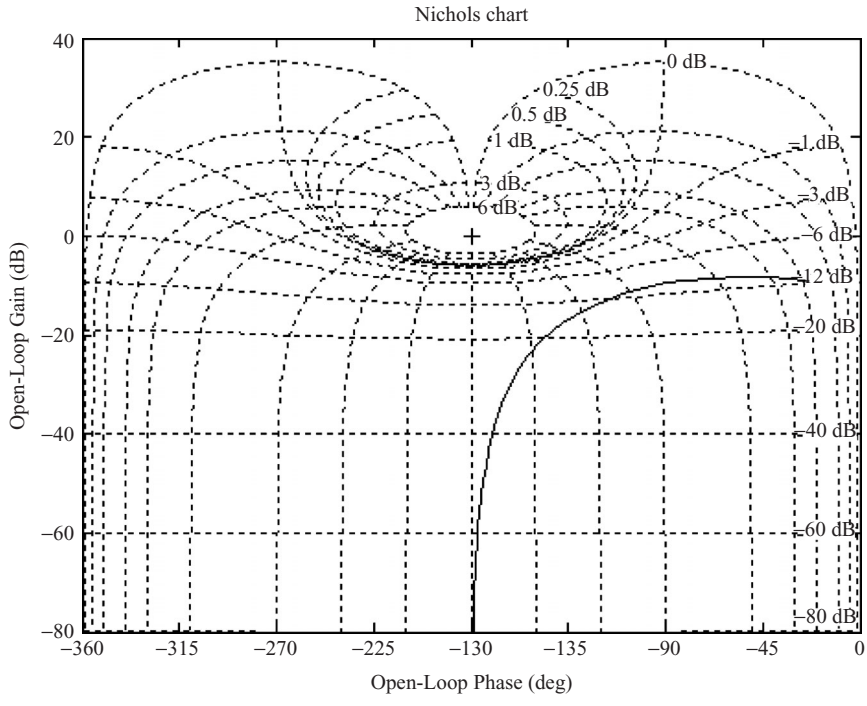

Figure B.5 Nichols diagram

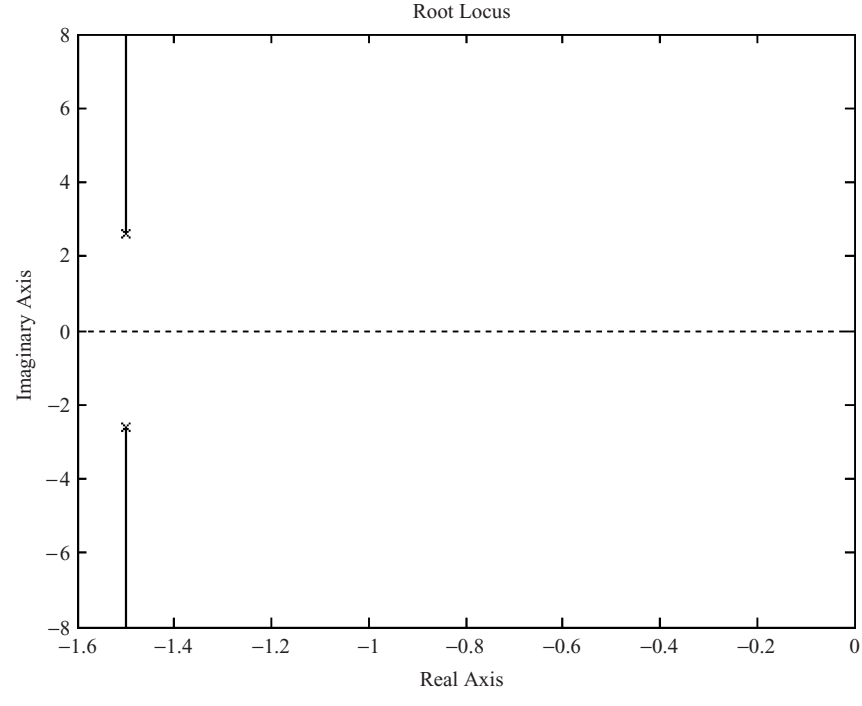

Figure B.6 Root locus diagram

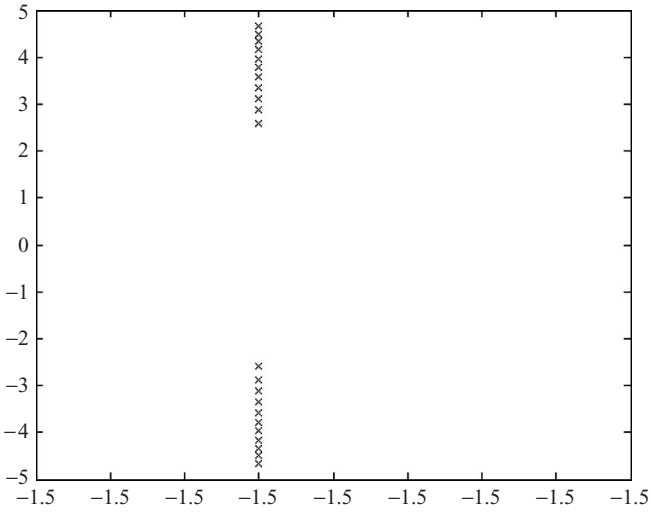

Figure B.7 Root locus diagram

```
p =
    -1.5000 + 2.5981i
    -1.5000 - 2.5981i

k =
    3
```

where z is the zeros, p is the poles and k is the gain.

Closed-loop system transfer function. The closed-loop system transfer function can be obtained using the *feedback* command. This command assumes by default a system a system with unity gain negative feedback. The closed-loop transfer function of the above system is thus given by

```
>> G = tf(num,den);
>> sys = feedback(G,1)
```

Transfer function:

```
       3
-----------------
s^2 + 3s + 12
```

Series and parallel connected transfer functions. Consider two serially connected transfer functions $G(s)H(s)$. The overall transfer function can be obtained as

```
series(G,H)
```

where G and H are the transfer functions $G(s)$ and $H(s)$, respectively, in MATLAB representation. For example, if

$$G(s) = \frac{1}{s^2 + 3s + 4} \quad \text{and} \quad H(s) = \frac{2}{s + 5}$$

then $G(s)H(s)$ can be obtained from

```
>> G = tf(1,[1  3  4]);
>> H = tf(2,[1  5]);
>> GH = series(G,H)
```

Transfer function:

```
             2
-------------------------
s^3 + 8 s^2 + 19 s + 20
```

Similarly, the *parallel* statement can be used to reduce the transfer functions connected in parallel.

Factored transfer functions. If a transfer function is in factored form, it can be entered using the *conv* command. For example, if

$$G(s) = \frac{(s + 4)}{(s + 1)(s + 2)}$$

then it can be entered into MATLAB as

```
>> num = [1  4];
>> den1 = [1  1];
>> den2 = [1  2];
>> den = conv(den1,den2);
```

Similarly, for the transfer function

$$G(s) = \frac{2}{(s + 1)(s + 2)(s + 4)}$$

we can write

```
>> num = 2;
>> den1 = [1  1];
>> den2 = [1  2];
>> den3 = [1  4];
>> den = conv(den1,conv(den2,den3));
```

Inverse Laplace transforms. The MATLAB command *residue* is used to obtain the inverse Laplace transform of a transfer function by finding the coefficients of the partial fraction expansion. The partial fraction expansion is assumed to be in the following format:

$$Y(s) = \frac{r(1)}{s - p(1)} + \frac{r(2)}{s - p(2)} + \frac{r(3)}{s - p(3)} + \cdots + \frac{r(n)}{s - p(n)} + k(s).$$

As an example, suppose that we wish to find the coefficients A, B and C of the partial fraction expansion

$$Y(s) = \frac{1}{(s+1)(s+2)(s+3)} = \frac{A}{s+1} + \frac{B}{s+2} + \frac{C}{s+3}.$$

The required MATLAB commands are

```
>> num = [1];
>> den = conv([1 1],conv([1 2],[1 3]));
>> [r,p,k] = residue(num,den)
```

r =

```
   0.5000
   -1.0000
   0.5000
```

p =

```
   3.0000
   -2.0000
   -1.0000
```

k =

```
   [ ]
```

The required partial fraction expansion is then

$$Y(s) = \frac{0.5}{s+1} - \frac{1}{s+2} + \frac{0.5}{s+3}.$$

B.2.2 Discrete-Time Systems

The Control System Toolbox also supports the design and analysis of discrete-time systems. Some of the most commonly used discrete-time system commands and algorithms are given in this section.

Discretizing a continuous transfer function. The C2d function can be used to discretize a continuous system transfer function. The sampling time must be specified. The default method of discretization is zero-order hold at the inputs, but other methods such as linear interpolation or bilinear approximation can be selected. For example, consider the continuous-time system transfer function

$$G(s) = \frac{1}{s+4}.$$

Assuming the sampling period is 0.1 s we can convert the transfer function to discrete time using the following commands:

```
>> G = tf(1, [1,4]);
>> Gz = c2d(G, 0.1)
```

Transfer function:

```
  0.08242
-----------
z - 0.6703

Sampling time: 0.1
```

Thus, the required discrete time transfer function is

$$G(z) = \frac{0.08242}{z - 0.6703}.$$

In the following example we convert a second-order continuous-time system,

$$G(s) = \frac{4}{s^2 + 4s + 2},$$

to discrete form, with sampling time 1 s:

```
>> G = tf(4,[1 4 2]);
>> Gz = c2d(G, 1)
```

Transfer function:

```
    0.6697 z + 0.1878
--------------------------
z^2 - 0.5896 z + 0.01832

Sampling time: 1
```

Poles and zeros. The poles and zeros can be obtained as follows:

```
>> [z,p,k] = zpkdata(Gz,'v')

z =
   -0.2804

p =
    0.5567
    0.0329

k =
    0.6697
```

Thus, $G(z)$ has one zero at -0.2804 and two poles at 0.5567 and 0.0329. The d.c. gain is 0.6697.

The positions of the poles and zeros can be plotted on the complex plane using the command

```
>> pzmap(num,den)
```

Also, the positions of the poles and zeros and the unit circle in the z-plane can be plotted using the command

```
>> zplane(num,den)
```

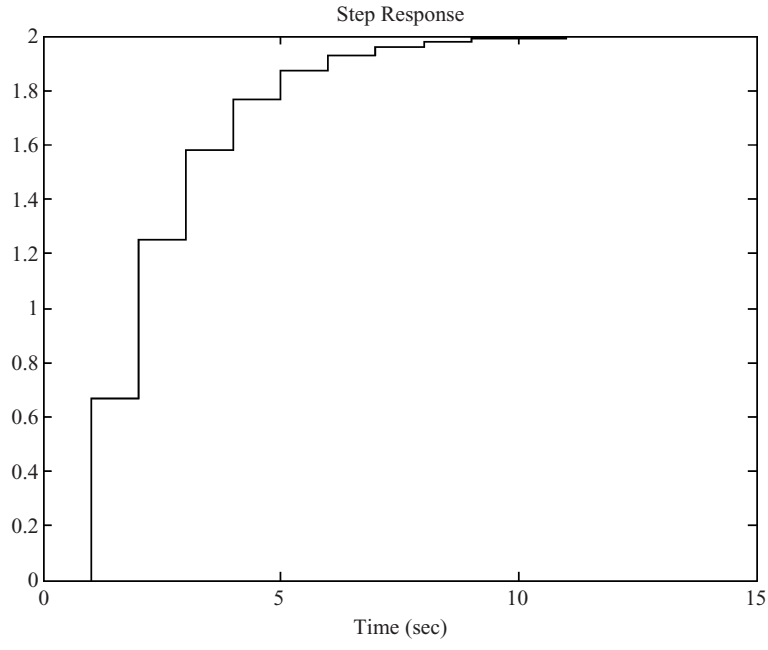

Figure B.8 Step response

Step response. The unit step response of $G(z)$ is obtained from

```
>> num = [0  0.6697  0.1878];
>> den = [1  -0.5896  0.01832];
>> dstep(num,den)
```

and the response obtained is shown in Figure B.8.

Impulse response. The impulse response of $G(z)$ is obtained by writing

```
>> num = [0  0.6697  0.1878];
>> den = [1  -0.5896  0.01832];
>> dimpulse(num,den)
```

and the response is shown in Figure B.9.

Root locus. The root locus diagram with lines of constant damping factor and lines of constant natural frequency is shown in Figure B.10 and is obtained from

```
>> zgrid('new');
>> rlocus(num,den)
```

The gain and the roots at any point on the locus can interactively be found using the command

```
>> [k,p] = rlocfind(num,den)
```

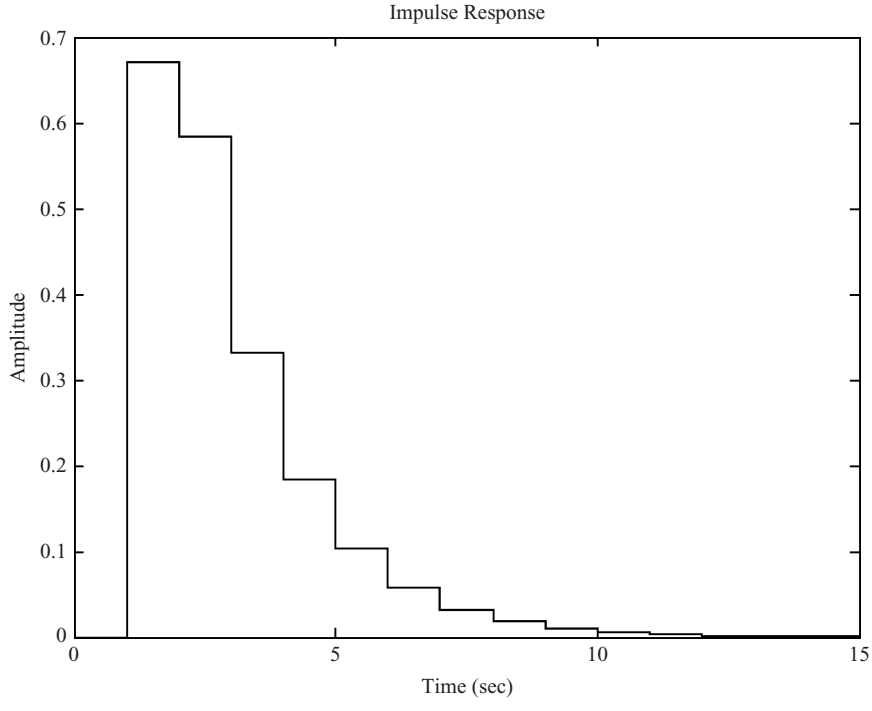

Figure B.9 Impulse response

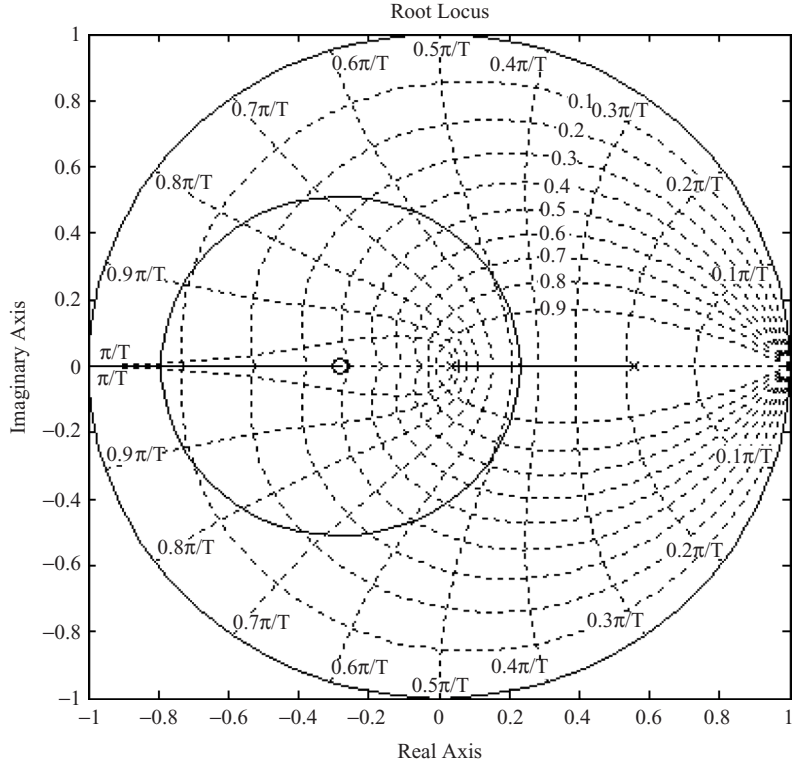

Figure B.10 Root locus diagram

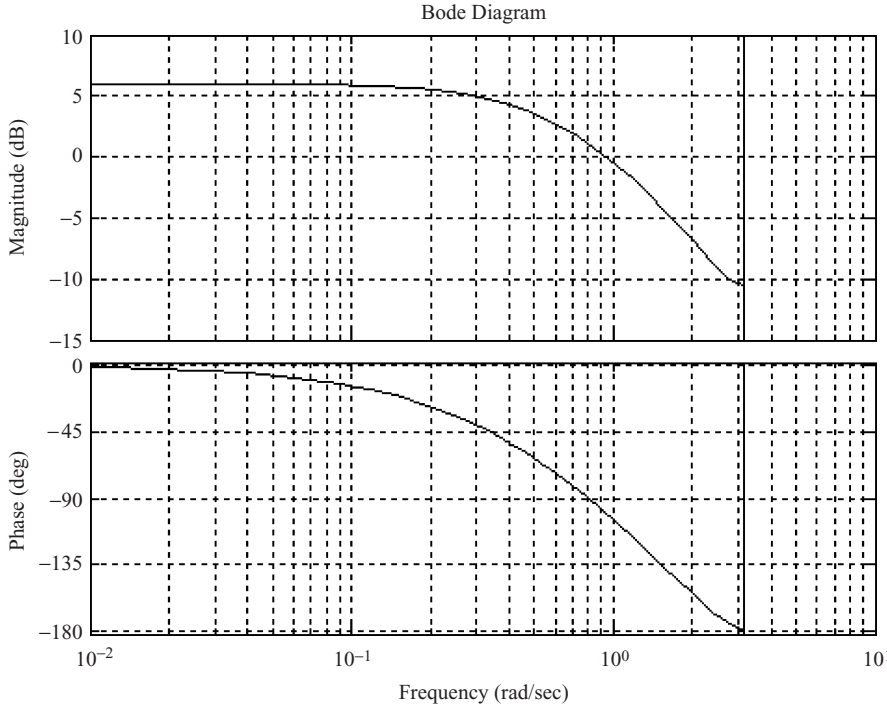

Figure B.11 Bode diagram

Bode diagram. The Bode diagram of a discrete time system can be obtained (assuming a sampling time of 1 s) as

```
>> dbode(num,den,1);
>> grid
```

The graph obtained is shown in Figure B.11.

Nyquist diagram. The Nyquist diagram of a discrete time system can be obtained as (assuming a sampling time of 1 s)

```
>> dnyquist(num,den,1);
```

The graph obtained is shown in Figure B.12.

z-Transform. The z-transform of a time function can be found using the MATLAB function *ztrans*. Some examples are given below.

The z-transform of $f(kT) = kT$ is found as

```
>> syms k T;
>> ztrans(k*T)

ans =
    T*z/(z-1)^2
```

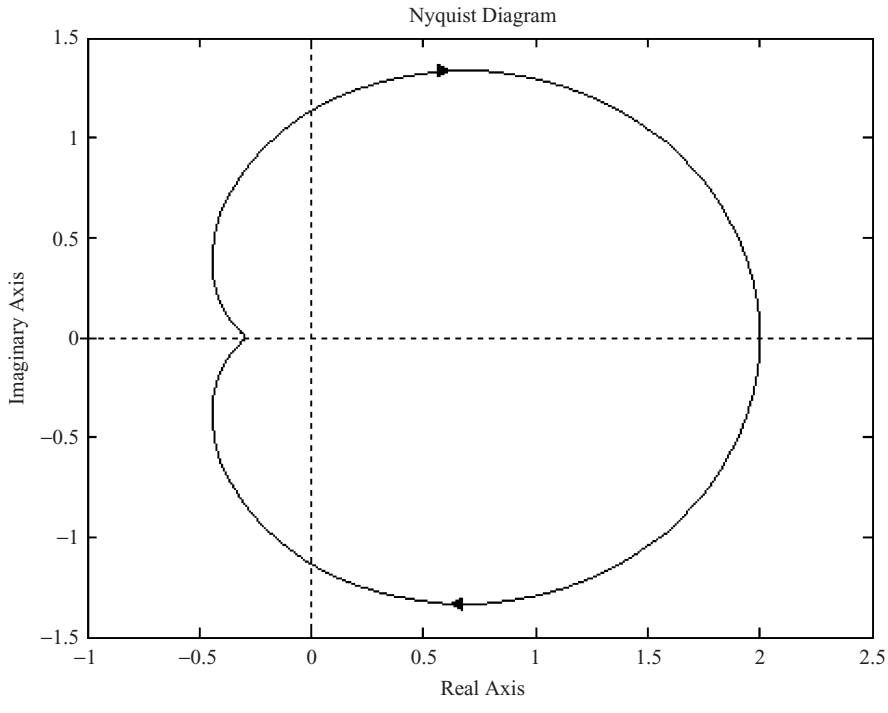

Figure B.12 Nyquist diagram

Notice that k and T are defined as symbols.

Similarly, the z-transform of $f(kT) = \sin(akT)$ is found as follows:

```
>> syms a k T;
>> f = sin(a*k*T);
>> ztrans(f)

ans =

    z*sin(a*T)/(z^2-2*z*cos(a*T)+1)
```

or

```
>> pretty(ans)

z sin(a T)
----------------------
2
z - 2 z cos(a T) + 1
```

Inverse z-transform. The inverse z-transform of a function can be found using the *iztrans* function. Some examples are given below.

The inverse z-transform of $F(z) = Tz/(z-1)^2$ is obtained as follows:

```
>> f = T*z/(z-1)^2;
>> iztrans(f)

ans =
    T*n
```

Notice that the default independent variable is n.

Coefficients of partial fraction expansion. MATLAB can be used to determine the coefficients of the partial fraction expansion. Some examples are given below.

Consider the transfer function

$$G(z) = \frac{2z^2 - z}{z^2 - 3z + 2}.$$

We usually expand the term $G(z)/z$ which gives a form which is usually easy to look up in the inverse z-transform tables. Thus,

$$\frac{G(z)}{z} = \frac{2z - 1}{z^2 - 3z + 2}.$$

The coefficients of the partial fraction expansion are found as follows:

```
>> [r,p,k] = residue([2 -1], [1  -3  2])

r =
   3
   -1

p =
   2
   1

k =
   []
```

where r are the residues, p are the poles and k are the direct terms. Thus,

$$\frac{G(z)}{z} = \frac{3}{z-2} - \frac{1}{z-1}$$

and

$$G(z) = \frac{3z}{z-2} - \frac{z}{z-1}$$

The time function can easily be found using z-transform tables.

Another example is given below where there is one direct term. Consider the transfer function

$$\frac{G(z)}{z} = \frac{2z^2 + 2z - 1}{z^2 - 3z + 2}.$$

The coefficients are found from

```
>> [r,p,k] = residue([2  2  -1], [1  -3  2])

r =
   11
   -3
```

```
p =
   2
   1

k =
   2
```

Thus,

$$\frac{G(z)}{z} = \frac{11}{z-2} - \frac{3}{z-1} + 2$$

or

$$G(z) = \frac{11z}{z-2} - \frac{3z}{z-1} + 2z$$

and the inverse z-transform can be found using z-transform tables.

The following example has a double pole. Consider the transfer function

$$\frac{G(z)}{z} = \frac{z^2 + 4z - 1}{z^3 - 5z^2 + 8z - 4}.$$

The coefficients are found from

```
>> [r,p,k] = residue([0  1  4  -1], [1  -5  8  -4])

r =
   -3.0000
   11.0000
   4.0000

p =
   2.0000
   2.0000
   1.0000

k =
   [ ]
```

There are two poles at $z = 2$, and this implies that there is a double root. The first residue is for the first-order term for the double root, and the second residue is for the second-order term for the double root. Thus,

$$\frac{G(z)}{z} = \frac{-3}{z-2} + \frac{11}{(z-2)^2} + \frac{4}{z-1}$$

or

$$G(z) = -\frac{3z}{z-2} + \frac{11z}{(z-2)^2} + \frac{4z}{z-1}.$$

The MATLAB command *residuez* can be used to compute the partial fraction expansion when the transfer function is written in powers of z^{-1}.