Section 6

Mathematical Functions and Applications

Outline

- Signals
- Polynomials
- Partial fraction expansion
- Functions of two variables
- User-defined functions
- Plotting functions

6.1 Signal Representation, Processing, and Plotting

As we have seen, one application of a one-dimensional array in MATLAB is to represent a function by its uniformly spaced samples. One example of such a function is a signal, which represents a physical quantity such as voltage or force as a function of time, denoted mathematically as $x(t)$. The value of x is known generically as the signal **amplitude**.

Sinusoidal Signal

A sinusoidal signal is a **periodic signal**, which satisfies the condition

$$x(t) = x(t + nT), \qquad n = 1, 2, 3, \ldots$$

where T is a positive constant known as the **period**. The smallest non-zero value of T for which this condition holds is the **fundamental period** T_0. Thus, the amplitude of the signal repeats every T_0.

Consider the signal

$$x(t) = A\cos(\omega t + \theta)$$

with **signal parameters**:

- A is the amplitude, which characterizes the peak-to-peak swing of $2A$, with units of the physical quantity being represented, such as volts.
- t is the independent time variable, with units of seconds (s).
- ω is the angular frequency, with units of radians per second (rad/s), which defines the fundamental period $T_0 = 2\pi/\omega$ between successive positive or negative peaks.
- θ is the initial phase angle with respect to the time origin, with units of radians, which defines the time shift $\tau = -\theta/\omega$ when the signal reaches its first peak.

With τ so defined, the signal $x(t)$ may also be written as

$$x(t) = A\cos\omega(t - \tau)$$

When τ is positive, it is a "time delay," that describes the time (greater than zero) when the first peak is reached. When τ is negative, it is a "time advance" that describes the time (less than zero) when the last peak was achieved. This sinusoidal signal is shown in Figure 6.1.

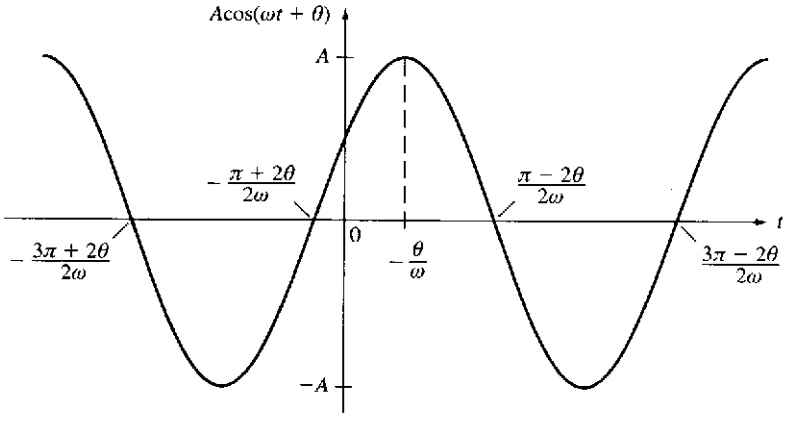

Figure 6.1: Sinusoidal signal $A\cos(\omega t + \theta)$ with $-\pi/2 < \theta < 0$.

Consider computing and plotting the following cosine signal

$$x(t) = 2\cos 2\pi t$$

Identifying the parameters:

- Amplitude $A = 2$
- Frequency $\omega = 2\pi$. The fundamental period $T_0 = 2\pi/\omega = 2\pi/2\pi = 1s$.
- Phase $\theta = 0$.

A time-shifted version of this signal, having the same amplitude and frequency is

$$y(t) = 2\cos[2\pi(t - 0.125)]$$

where the time shift $\tau = 0.125s$ and the corresponding phase is $\theta = 2\pi(-0.125) = -\pi/4$.

A third signal is the related sine signal having the same amplitude and frequency

$$z(t) = 2\sin 2\pi t$$

Preparing a script to compute and plot $x(t)$, $y(t)$, and $z(t)$ over two periods, from $t = -1s$ to $t = 1s$:

```
% sinusoidal representation and plotting
%
t = linspace(-1,1,101);
x = 2*cos(2*pi*t);
y = 2*cos(2*pi*(t-0.125));
z = 2*sin(2*pi*t);
plot(t,x,t,y,t,z),...
  axis([-1,1,-3,3]),...
  title('Sinusoidal Signals'),...
  ylabel('Amplitude'),...
  xlabel('Time (s)'),...
  text(-0.13,1.75,'x'),...
  text(-0.07,1.25,'y'),...
  text(0.01,0.80,'z'),grid
```

Thus, `linspace` has been used to compute the time row vector `t` having 101 samples or elements with values from -1 to 1. The three signals are computed as row vectors and plotted, with axis control, labels and annotation. The resulting plot is shown in Figure 6.2. Observe that $x(t)$ reaches its peak value of 2 at time $t = 0$, which we can verify as being correct since we know $\cos 0 = 1$. The signal $y(t)$ is $x(t)$ delayed by $\tau = 0.125s$. The signal $z(t)$ is 0 for $t = 0$, since $\sin 0 = 0$. It reaches its first peak at $t = T_0/4 = 0.25s$, as $\sin(2\pi \cdot 0.25) = \sin(\pi/2) = 1$.

Phasor Representation of a Sinusoidal Signal

An important and very useful representation of a sinusoidal signal is as a function of a complex exponential signal.

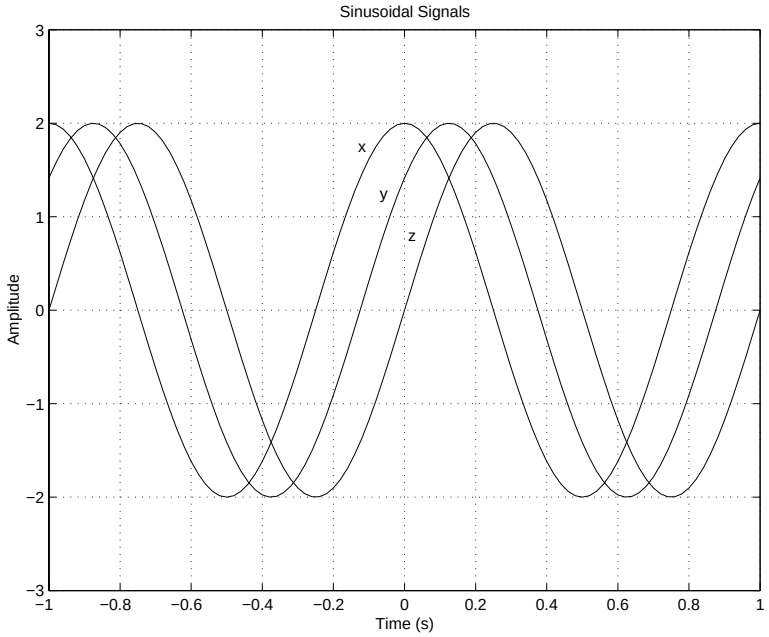

Figure 6.2: Sinusoidal signals

Recall

$$e^{j\theta} = \cos\theta + j\sin\theta$$

Thus,

$$\cos\theta = \mathrm{Re}\left[e^{j\theta}\right]$$

As a result, the signal

$$x(t) = A\cos(\omega t + \phi)$$

can be *represented* as the real part of the complex exponential

$$\begin{aligned} x(t) &= \mathrm{Re}\left[Ae^{j(\omega t+\phi)}\right] \\ &= \mathrm{Re}\left[Ae^{j\phi}e^{j\omega t}\right] \end{aligned}$$

We call $Ae^{j\phi}e^{j\omega t}$ the complex representation of $x(t)$ and write

$$x(t) \longleftrightarrow Ae^{j\phi}e^{j\omega t}$$

meaning that the signal $x(t)$ may be reconstructed by taking the real part of $Ae^{j\phi}e^{j\omega t}$. In this representation, we call $Ae^{j\phi}$ the *phasor* or complex amplitude representation of $x(t)$ and write

$$x(t) \longleftrightarrow Ae^{j\phi}$$

meaning that the signal $x(t)$ may be reconstructed from $Ae^{j\phi}$ by multiplying by $e^{j\omega t}$ and taking the real part.

The phasor representation of the sinusoid $x(t) = A\cos(\omega t+\phi)$ is shown in the complex plane in Figure 6.3. At $t = 0$, the complex representation produces the phasor $Ae^{j\phi}$, where ϕ is approximately

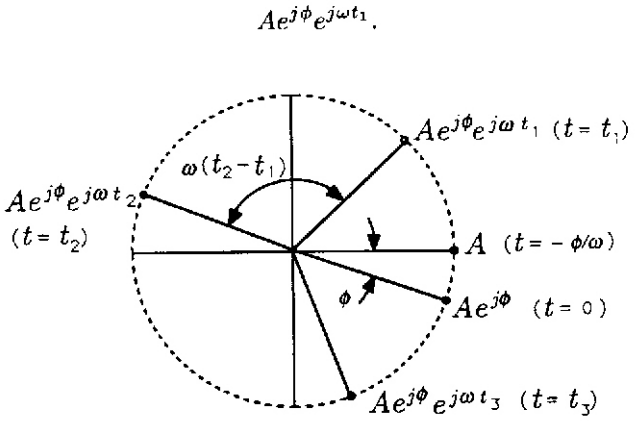

Figure 6.3: Rotating phasor

$-\pi/10$. If time t increases to time t_1, then the complex representation produces

$$Ae^{j\phi}e^{j\omega t_1}$$

From our discussion of complex numbers, we know that $e^{j\omega t_1}$ rotates the phasor $Ae^{j\phi}$ through an angle ωt_1. Therefore, as we increase t from 0, we rotate the phasor from $Ae^{j\phi}$, producing the circular trajectory around the circle of radius A shown in Figure 6.3. When $t = 2\pi/\omega$, then $e^{j\omega t} = e^{j\pi} = 1$. Therefore, every $2\pi/\omega$ seconds, the phasor revisits any given position on the circle. The quantity $Ae^{j\phi}e^{j\omega t}$ is called a *rotating phasor* whose rotation rate is the frequency ω:

$$\frac{d}{dt}\omega t = \omega$$

The rotation rate is also the frequency of the sinusoidal signal $x(t) = A\cos(\omega t + \phi)$.

The real part of the complex, or rotating phasor, representation $Ae^{j\phi}e^{j\omega t}$ is the desired signal $x(t) = A\cos(\omega t + \phi)$. This real part is read off of the rotating phasor diagram as shown in Figure 6.4.

Consider computing and plotting the phasor

$$x(t) = e^{j\omega t}$$

where $\omega = 2000\pi$ rad/s and t ranges from -2×10^{-3} s to 2×10^{-3} s, in steps of 0.02×10^{-3} s. Also to be plotted are $y(t) = \mathrm{Re}[x(t)]$ and $z(t) = \mathrm{Im}[x(t)]$. The script file is

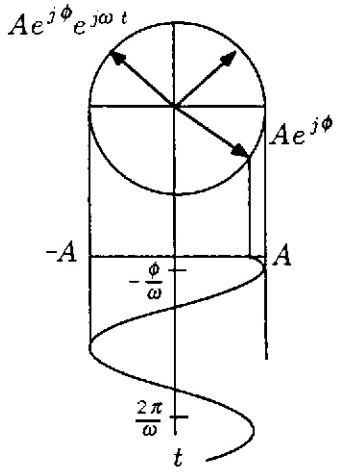

Figure 6.4: Reading a real signal from a complex, rotating phasor

```
% Phasor computation and plot
%
t = (-2e-03:0.02e-03:2e-03);
x = exp(j*2000*pi*t);
y = real(x);
z = imag(x);
subplot(2,1,1),plot(x,':'),...
   axis square,...
   title('exp(jwt)'),...
   xlabel('Real'),...
   ylabel('Imaginary'),...
subplot(2,1,2),plot(t,y,'-',t,z,':'),...
   title('Re[exp(jwt)] and Im[exp(jwt)] vs t   w=1000*2*pi'),...
   xlabel('Time (s)'),grid on,...
   legend('Re[exp(j \omega t)]','Im[exp(j \omega t)]',-1)
```

and the resulting plot is shown in Figure 6.5, where $y(t)$ is plotted with a solid line and $z(t)$ is plotted with a dotted line.

Harmonic Motion

A periodic motion that can be described by a sinusoidal function of time is called **harmonic motion**. A mechanism that produces such motion is a *scotch yoke*, shown in Figure 6.6. This mechanism is used in machines known as *shakers*, for testing the behavior of equipment subject to vibrations. The driving element is a rotating disk with a pin mounted a distance A from the center. The pin can slide in the slot of the element marked *x-yoke*. The motion of the *x-yoke* is restricted by a guided rod attached to it, so that this yoke can move only horizontally. A similar slotted element, marked *y-yoke*, is assembled above the *x-yoke*. The motion of the *y-yoke* is restricted by a guided rod that allows only vertical displacements.

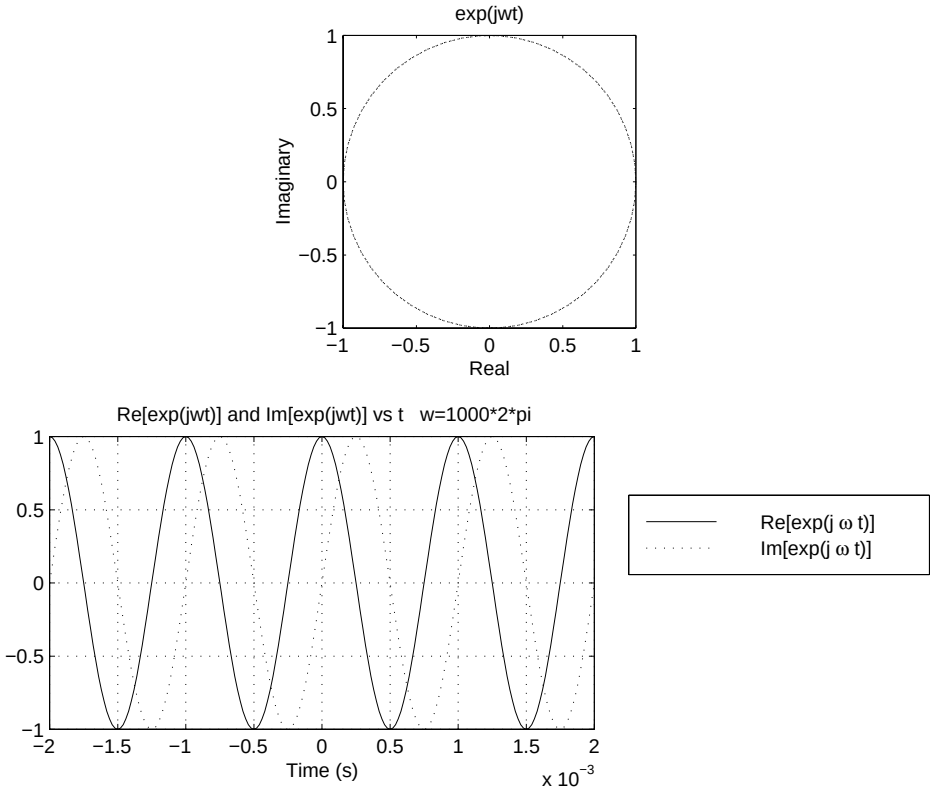

Figure 6.5: The signals $e^{j\omega t}$, Re $[e^{j\omega t}]$, and Im $[e^{j\omega t}]$

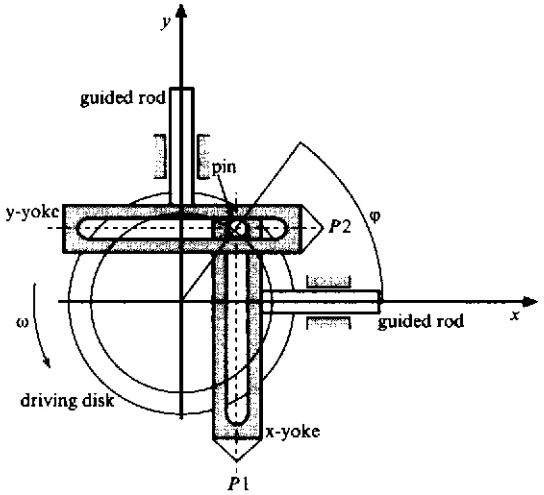

Figure 6.6: Scotch yoke mechanism

The motion is described mathematically by first assuming that at time $t = 0$ the position of the pin relative to the x axis is defined by the angle ϕ. The x-coordinate of the point $P1$ is

$$x(0) = A\cos(\phi)$$

and the y-coordinate of the point $P2$ is

$$y(0) = A\sin(\phi)$$

If the angular speed of the disk is ω radians per second, then the x-coordinate of the point $P1$ at time t is

$$x(t) = A\cos(\omega t + \phi)$$

and the y-coordinate of the point $P2$ is

$$y(t) = A\sin(\omega t + \phi)$$

Using the phasor representation for this motion, we can define

$$z(t) = Ae^{j\omega t+\phi}$$

and we observe that $x(t)$ is the real part and $y(t)$ is the imaginary part of $z(t)$. The point z can be considered to be the end of a vector with origin at the coordinate origin and magnitude A. This vector rotates with angular velocity ω, starting from angle ϕ.

If $y(t)$ is the **displacement** of the harmonic motion, the **velocity** is

$$v(t) = \frac{dy}{dt} = \omega A\cos(\omega t + \phi) = \omega A\sin(\omega t + \phi + \pi/2)$$

and the **acceleration** is

$$a(t) = \frac{d^2y(t)}{dt^2} = -\omega^2 A\sin(\omega t + \phi) = \omega^2 A\sin(\omega t + \phi + \pi)$$

We conclude that the velocity of the harmonic motion can be represented by a rotating vector leading the displacement by the phase $\pi/2$, and the acceleration can be represented by another rotating vector, leading the displacement by the phase angle π.

Phasor Properties

Positive and Negative Frequencies

An alternative phasor representation for the signal

$$x(t) = A\cos(\omega t + \phi)$$

is obtained by using the Euler identity

$$\cos\theta = \frac{1}{2}\left(e^{j\theta} + e^{-j\theta}\right)$$

which yields

$$\begin{aligned} x(t) &= \frac{A}{2}\left[e^{j(\omega t+\phi)} + e^{-j(\omega t+\phi)}\right] \\ &= \frac{A}{2}e^{j\phi}e^{j\omega t} + \frac{A}{2}e^{-j\phi}e^{-j\omega t} \end{aligned}$$

In this equation, the term $\frac{A}{2}e^{j\phi}e^{j\omega t}$ is a rotating phasor that begins at the phasor value $\frac{A}{2}e^{j\phi}$ and rotates counterclockwise with frequency ω. The term $\frac{A}{2}e^{-j\phi}e^{-j\omega t}$ is a rotating phasor that begins at the (complex conjugate) phasor value $\frac{A}{2}e^{-j\phi}$ (for $t = 0$) and rotates clockwise with (negative) frequency ω. The physically meaningful frequency for a cosine is ω, a positive quantity. A negative frequency is not physically meaningful, but just means that the direction of rotation for the rotating phasor is clockwise, not counterclockwise, in the complex exponential representation of the real sinusoid. Thus, the concept of negative frequency is just an artifact of the two-phasor representation. In the one-phasor representation, when we take the "real part," the artifact does not arise. You are likely to encounter both the one-phasor and two-phasor representations, so you should be familiar with both.

Adding Phasors

The sum of two signals with common frequencies but different amplitudes and phases is

$$A_1\cos(\omega t + \phi_1) + A_2\cos(\omega t + \phi_2).$$

The rotating phasor representation for this sum is

$$\left(A_1e^{j\phi_1} + A_2e^{j\phi_2}\right)e^{j\omega t}$$

The new phasor is

$$A_1e^{j\phi_1} + A_2e^{j\phi_2}$$

and the corresponding real signal is

$$x(t) = \mathrm{Re}\left[\left(A_1e^{j\phi_1} + A_2e^{j\phi_2}\right)e^{j\omega t}\right]$$

The new phasor is shown in Figure 6.7, where the parallelogram rule for adding complex numbers applies.

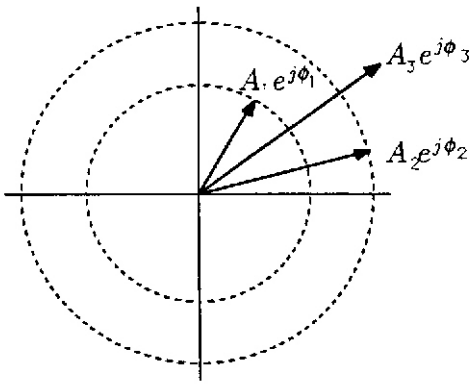

Figure 6.7: Adding phasors

Beating Between Tones

If you have heard two slightly mistuned musical instruments playing pure tones whose frequencies were close but not equal, you have sensed a beating phenomenon in which you perceive a single pure tone whose amplitude slowly varies periodically. The single perceived tone can be shown to have a frequency that is the average of the two mismatched frequencies, amplitude modulated by a tone whose "beat" frequency is half the difference between the two mismatched frequencies. This effect is shown in Figure 6.8.

To understand this phenomenon, begin with two pure tones whose frequencies are $\omega_0 + \omega_b$ and $\omega_0 - \omega_b$ (for example, $\omega_0 = 2\pi \times 1400$ rad/s and $\omega_b = 2\pi \times 100$ rad/s). The average frequency is ω_0 and the difference frequency is $2\omega_b$. You hear the sum of the two tones:

$$x(t) = A_1 \cos\left[(\omega_0 + \omega_b)t + \phi_1\right] + A_2 \cos\left[(\omega_0 - \omega_b)t + \phi_2\right]$$

Assume that the amplitudes are equal, with $A = A_1 = A_2$. The phases may be written as

$$\phi_1 = \phi + \psi \quad \text{and} \quad \phi_2 = \phi - \psi$$

with

$$\phi = \frac{1}{2}(\phi_1 + \phi_2) \quad \text{and} \quad \psi = \frac{1}{2}(\phi_1 - \phi_2)$$

Representing $x(t)$ as a complex phasor

$$\begin{aligned} x(t) &= A \,\mathrm{Re}\left\{e^{j[(\omega_0+\omega_b)t+\phi+\psi]} + e^{j[(\omega_0-\omega_b)t+\phi-\psi]}\right\} \\ &= A \,\mathrm{Re}\left\{e^{j(\omega_0 t+\phi)}\left[e^{j(\omega_b t+\psi)} + e^{-j(\omega_b t+\psi)}\right]\right\} \\ &= 2A \,\mathrm{Re}\left\{e^{j(\omega_0 t+\phi)}\cos(\omega_b t+\psi)\right\} \\ &= 2A\cos(\omega_0 t+\phi)\cos(\omega_b t+\psi) \end{aligned}$$

This is an amplitude modulated waveform, in which a low frequency signal with beat frequency ω_b modulates a high frequency signal with carrier frequency ω_0 rad/s. Over short periods of time, the modulating signal $\cos(\omega_b t + \psi)$ remains relatively constant while the carrier term $\cos(\omega_0 + \phi)$ produces many cycles of its pure tone. Thus, we perceive the pure tone at the carrier frequency ω_0, having an amplitude that varies sinusoidally at the beat frequency ω_b.

The following is the script to simulate beating tones in which $\omega_0 = 2\pi \times 1400$ rad/s and $\omega_b = 2\pi \times 100$ rad/s, resulting in the plot shown in Figure 6.8.

```
% beating sinusoidal tones
%
t = linspace(-1e-2,1e-2,1001);
x = cos(2*pi*1500*t) + cos(2*pi*1300*t);
m = 2*cos(2*pi*100*t);
plot(t,m,'b:',t,-m,'b:',t,x,'k'),...
  axis([-0.01 0.01 -2.4 2.4]),...
  title('Beating between tones'),...
  xlabel('Time (s)'),...
  ylabel('Amplitude')
```

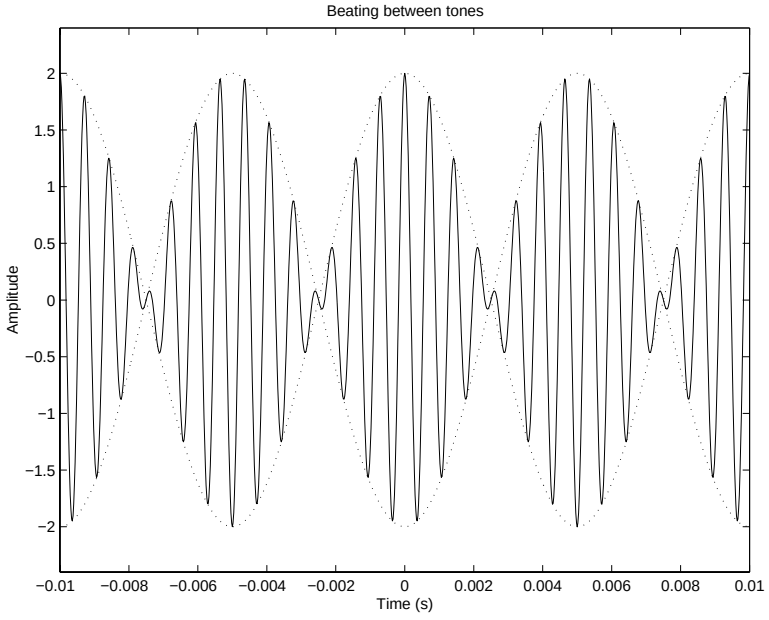

Figure 6.8: Beating between tones

6.2 Polynomials

A **polynomial** is a function of a single variable that can be expressed in the general form

$$A(s) = a_1 s^N + a_2 s^{N-1} + a_3 s^{N-2} + \cdots + a_N s + a_{N+1}$$

where the variable is s and the **polynomial coefficients** are the $N+1$ constants $a_1, a_2, \ldots, a_{N+1}$. The polynomial is of **degree** N, the largest value used as an exponent. The general form of a degree 3 (cubic) polynomial is

$$A(s) = a_1 s^3 + a_2 s^2 + a_3 s + a_4$$

and a specific example of a cubic polynomial is

$$A(s) = s^3 + 4s^3 - 7s - 10$$

Note that the notation used here is nonstandard, as the coefficient of term s^k is usually denoted as a_k. However, the nonstandard notation is more compatible with the indexing of arrays in MATLAB, as will be explained below.

For information on MATLAB functions supporting polynomial computations, type `help polyfun`.

Polynomial Evaluation

There are several ways to evaluate a polynomial for a set of values. Consider the cubic polynomial:

$$A(s) = s^3 + 4s^2 - 7s - 10$$

- Scalar `s`: use scalar operations

  ```
  A = s^3 + 4*s^2 - 7*s - 10;
  ```

- Vector or matrix `s`: use array or element-by-element operations:

  ```
  A = s.^3 + 4*s.^2 - 7*s - 10;
  ```

  The size of the vector or matrix `A` will be the same as that of `s`.

- Using `polyval(a,s)`: Evaluates a polynomial with coefficients in vector `a` for the values in `s`. The result is a matrix the same size as `s`. Element `a(1)` corresponds to coefficient a_1.

Consider evaluating and plotting $A(s)$ over the interval [-1,3]:

```
s = linspace(-1,3,201);
a = [1 4 -7 -10];
A = polyval(a,s);
plot(s,A),...
  title('Polynomial Function A(s) = s^3 + 4s^2 -7s -10'),...
  xlabel('s'),...
  ylabel('A(s)')
```

The resulting plot is shown in Figure 6.9.

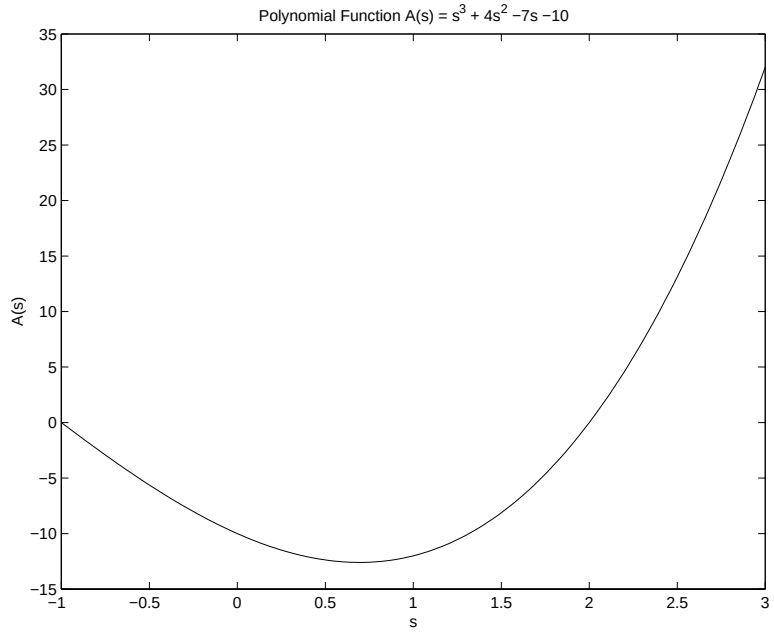

Figure 6.9: Plot of polynomial function $A(s)$

Polynomial Operations

By characterizing polynomial $A(s)$ by coefficients a_k stored in vector `a` and polynomial $B(s)$ by coefficients b_k stored in vector `b`, algebraic operations can be performed on the two polynomials.

- **Addition:** The coefficients of the sum of two polynomials is the sum of the coefficients of the two polynomials. The vectors containing the coefficients must be of the same length. For example, to add

$$A(s) = s^4 - 3s^3 - s + 2$$

$$B(s) = 4s^3 - 2s^2 + 5s - 16$$

$$\begin{aligned} C(s) &= A(s) + B(s) \\ &= s^4 + (4-3)s^3 - 2s^2 + (5-2)s + (2-16) \\ &= s^4 + s^3 - 2s^2 + 4s - 14 \end{aligned}$$

The script to perform this addition:

```
>> a = [1 -3 0 -1 2];
>> b = [0 4 -2 5 -16];
>> c = a + b
c =
     1     1    -2     4   -14
```

Thus,

$$C(s) = s^4 + s^3 - 2s^2 + 4s - 14$$

- **Scalar multiple:** The coefficient vector of the scalar multiple of a polynomial is simply the scalar times the coefficient vector of the polynomial. To specify

  $$C(s) = 3A(s)$$

  for $A(s)$ above, the following commands are used:

```
>> a = [1 -3 0 -1 2];
>> c = 3*a
c =
     3    -9     0    -3     6
```

  Thus

  $$C(s) = 3s^4 - 9s^3 - 3s + 6$$

- **Multiplication:** Apply the function `conv(a,b)`, which returns the coefficient vector for the polynomial resulting from the product of polynomials represented by the coefficients in `a` and `b`. The vectors `a` and `b` don't have to be of the same length. The resulting vector has length equal to the sum of the lengths of `a` and `b` minus one.

  Consider the following polynomial product, evaluated by hand using the "first-outer-inner-last" (FOIL) method:

  $$F(s) = (s+2)(s+3) = s^2 + 3s + 2s + 6 = s^2 + 5s + 6$$

  Using a MATLAB script

```
>> a = [1 2];
>> b = [1 3];
>> c = conv(a,b)
c =
     1     5     6
```

  Now consider multiplying polynomials $A(s)$ and $B(s)$ above:

  $$C(s) = A(s)B(s) = (s^4 - 3s^3 - s + 2)(4s^3 - 2s^2 + 5s - 16)$$

  Recall from your algebra class that evaluation by hand is a tedious process. Using a MATLAB script:

```
>> a = [1 -3 0 -1 2];
>> b = [4 -2 5 -16];
>> c = conv(a,b)
c =
     4   -14    11   -35    58    -9    26   -32
```

Thus,

$$C(s) = 4s^7 - 14s^6 + 11s^5 - 35s^4 + 58s^3 - 9s^2 + 26s - 32$$

- **Division:** You may have learned the "long division" method for evaluating the division of two polynomials. This method won't be reviewed here, except to remind you that two results are determined: the quotient and the remainder. The result, expressed mathematically, is

$$\frac{N(s)}{D(s)} = Q(s) + \frac{R(s)}{D(s)}$$

where $N(s)$ is the numerator polynomial, $D(s)$ is the denominator polynomial, $Q(s)$ is the quotient polynomial, and $R(s)$ is the remainder polynomial.

The MATLAB function to perform polynomial division:

`[q,r] = deconv(n,d)` Returns quotient polynomial coefficients `q` and remainder polynomial coefficients `r` from numerator coefficients `n` and denominator coefficients `d`.

Consider the evaluation of polynomial division in which the numerator is $C(s)$ and the denominator is $A(s)$, both from the multiplication example above. The result for the quotient should then be $B(s)$ above and the remainder should be all zeroes.

```
>> c = [4   -14    11   -35    58    -9    26   -32];
>> a =  [1 -3 0 -1 2];
>> [q r] = deconv(c,a)
q =
     4    -2     5   -16
r =
     0     0     0     0     0     0     0     0
```

Now consider the following:

$$H(s) = \frac{N(s)}{D(s)} = \frac{s^3 + 5s^2 + 11s + 13}{s^2 + 2s + 4}$$

In MATLAB:

```
>> n = [1 5 11 13];
>> d = [1 2 4];
>> [q r] = deconv(n,d)
q =
     1     3
r =
     0     0     1     1
```

Placing the result in mathematical form:

$$\begin{aligned} H(s) &= Q(s) + \frac{R(s)}{D(s)} \\ &= s + 3 + \frac{s+1}{s^2 + 2s + 4} \end{aligned}$$

- **Derivatives:** You should be familiar with the rule for differentiating a polynomial term

$$\frac{d}{ds}as^n = nas^{n-1}$$

Applying this rule to the polynomial

$$P(s) = s^3 + 4s^2 - 7s - 10$$

produces the derivative

$$\frac{d}{ds}P(s) = 3s^2 + 8s - 7$$

MATLAB provides function `polyder` for polynomial differentiation.

`polyder(p)`	Returns the coefficients of the derivative of the polynomial whose coefficients are the elements of vector p.
`polyder(a,b)`	Returns the coefficients of the derivative of the product polynomial $A(s) * B(s)$.
`[n,d] = polyder(b,a)`	Returns the derivative of the polynomial ratio $B(s)/A(s)$, represented as $N(s)/D(s)$.

Confirming the example above using the first form of `polyder`:

```
>> p = [1 4 -7 -10];
>> d = polyder(p)
d =
     3     8    -7
```

The second form of `polyder` returns the coefficients of the derivative of a product of two polynomials. This is equivalent to multiplying the two polynomials with `conv` and then differentiating using the first form of `polyder`. Consider applying the second form of `polyder` to the second example of polynomial multiplication above. The result can be confirmed by differentiating $C(s)$ above by hand.

```
>> a = [1 -3 0 -1 2];
>> b = [4 -2 5 -16];
>> dc = polyder(a,b)
dc =
    28   -84    55  -140   174   -18    26
```

The derivative of a polynomial ratio is more difficult to evaluate by hand. Using the derivative notation

$$f' = \frac{df}{ds},$$

recall the quotient rule for differentiation

$$\left(\frac{f}{g}\right)' = \frac{gf' - fg'}{g^2}$$

Thus, for the polynomial ratio

$$H(s) = \frac{s+2}{s+3}$$

the derivative is

$$\frac{dH(s)}{ds} = \frac{(s+3)-(s+2)}{(s+3)^2} = \frac{1}{(s+3)^2} = \frac{1}{s^2+6s+9}$$

Applying the third form of `polyder`

```
>> b = [1 2];
>> a = [1 3];
>> [q,d] = polyder(b,a)
q =
     1
d =
     1     6     9
```

Note that the denominator polynomial `d` in the result is in expanded, rather than factored form.

Roots of Polynomials

In many engineering problems, there is a need to find the **roots** of a polynomial $P(s)$, which are the values of s for which $P(s) = 0$. When $P(s)$ is of degree N, then there are exactly N roots, which may be **repeated roots** or **complex roots**. If the polynomial coefficients $(a_1, a_2, \ldots)$ are real, then any complex roots will always occur in complex conjugate pairs.

For degree one (linear) or two (quadratic), the roots are easily determined. The quadratic equation can be used for degree two, as described earlier. For polynomials of degree 3 and higher, numerical techniques are required to find the roots. The MATLAB function for finding the roots:

`roots(a)`	Returns as a vector the roots of the polynomial represented by the coefficient vector `a`.

Consider the denominator polynomial from the example above

$$D(s) = (s+3)^2 = s^2 + 6s + 9$$

```
>> d = [1 6 9];
>> roots(d)
ans =
    -3
    -3
```

This confirms that there are two roots at -3 in the denominator.

Consider the cubic polynomial

$$P(s) = s^3 - 2s^2 - 3s + 10$$

The commands to compute and display the roots:

```
>> p = [1,-2,-3,10];
>> r = roots(p)
r =
   2.0000+ 1.0000i
   2.0000- 1.0000i
  -2.0000
```

Note that there are 3 roots for the degree 3 polynomial, with a complex conjugate pair of roots and a real root. To verify that these values are roots, evaluate the polynomial at the roots:

```
>> P= polyval(p,r)
P =
  1.0e-013 *
  -0.0355+ 0.1377i
  -0.0355- 0.1377i
   0.0711
```

While $P(r)$ is not exactly zero, due to the limitations on numerical accuracy, for each root it is of the order of 10^{-14}.

The roots can be used to express the polynomial in factored form. For example:

$$P(s) = s^3 - 2s^2 - 3s + 10 = (s - r_1)(s - r_2)(s - r_3) = (s - 2 - j)(s - 2 + j)(s + 2)$$

The coefficients of the polynomial can be determined from the roots using the `poly` function:

`poly(r)`	Returns as a row vector the coefficients of the polynomial whose roots are contained in the vector `r`.

For example:

```
>> a = poly(r)
a =
    1.0000   -2.0000   -3.0000   10.0000
```

Example 6.1 *Finding the depth of a well using roots of a polynomial*

Consider the problem of finding the depth of a well by dropping a stone and measuring the time t to hear a splash. This time is composed of the time t_1 of free fall from release to reaching the water and the time t_2 that the sound takes to travel from the water surface to the ear of the person dropping the stone. Let g denote the acceleration of gravity, d the well depth (approximately equal to the distance between the hand or the ear of the person and the water surface), and c the speed of sound in air. The depth d, the distance traveled by the stone during time t_1 is

$$d = \frac{g}{2} \cdot t_1^2$$

or

$$t_1 = \sqrt{2d/g}$$

The same distance traveled by the sound during t_2 is

$$d = c \cdot t_2$$

or

$$t_2 = d/c$$

The total time is

$$t = t_1 + t_2 = \sqrt{2d/g} + d/c$$

Squaring the equation above and rearranging the terms

$$d^2 - 2(tc + c^2/g)d + c^2t^2 = 0$$

The depth d is the solution (roots) of the quadratic polynomial equation above. If the measured time t was 2.5s and the speed of sound c in air at atmospheric pressure and 20° Celsius is 343m/s, the following MATLAB script can be used to calculate the depth d.

```
% Well depth problem
%
% Define input values
t = 2.5;          % time to hear splash (s)
g = 9.81;         % acceleration of gravity (m/s^2)
c = 343;          % speed of sound in air (m/s)

% Calculate polynomial coefficients
a(1) = 1;
a(2) = -2*(t*c + c^2/g);
```

```
a(3) = (c*t)^2;

% Find roots corresponding to depth
depth = roots(a)
```

The displayed results from this script:

```
depth =
  1.0e+004 *
    2.5672
    0.0029
```

As is the case in many problems involving roots of a quadratic equation, one of the solutions is not physically reasonable. In this problem, the first root gives an impossibly large well depth, while the second root gives a reasonable depth. Investigating this solution with MATLAB:

```
>> d = depth(2)
d =
   28.6425
>> t1 = sqrt(2*d/g)
t1 =
    2.4165
>> t2 = d/c
t2 =
    0.0835
>> t = t1 + t2
t =
    2.5000
```

Thus, the depth is 28.6m, confirming the time of 2.5s.

■

6.3 Partial Fraction Expansion

A **rational function** is a ratio of polynomials having the form

$$H(s) = \frac{B(s)}{A(s)} = \frac{b_1 s^m + b_2 s^{m-1} + \cdots + b_m s + b_{m+1}}{a_1 s^n + a_2 s^{n-1} + \cdots + a_n s + a_{n+1}}$$

For $m < n$, $H(s)$ is known as a **proper rational function** and for $m \geq n$, it is known as an **improper rational function**. Denoting the roots of the denominator by $r_1, r_2, \ldots r_n$, $A(s)$ can be written in factored form as

$$A(s) = a_1(s - r_1)(s - r_2)\cdots(s - r_n)$$

and $H(s)$ can be written as

$$H(s) = \frac{B(s)}{a_1(s-r_1)(s-r_2)\cdots(s-r_n)}$$

Partial fraction expansion is a technique to express $H(s)$ as a sum of terms.

Expansion of Proper Rational Functions

For a proper rational function ($m < n$), there are three different cases of partial fraction expansion that must be considered:

1. Distinct (nonrepeated) real roots.
2. Distinct complex roots.
3. Repeated roots.

Distinct Real Roots

When the roots $r_1, r_2, \ldots, r_n$ are distinct and real, then by **partial fraction expansion** $H(s)$ can be expressed as

$$H(s) = \frac{c_1}{s-r_1} + \frac{c_2}{s-r_2} + \cdots + \frac{c_n}{s-r_n}$$

where the constants c_i are called the **residues**, which can be computed using the `residue` command:

`[c,r] = residue(b,a)` finds the residues `c` and roots `r` of a partial fraction expansion of the ratio of two polynomials $B(s)/A(s)$. Vectors `b` and `a` specify the coefficients of the numerator and denominator polynomials in descending powers of s. The residues are returned in the column vector `c` and the roots in column vector `r`.

Example:

$$H(s) = \frac{s+2}{s^3+4s^2+3s}$$

```
>> b = [1 2];
>> a = [1 4 3 0];
>> [c,r] = residue(b,a)
c =
   -0.1667
   -0.5000
    0.6667
```

```
r =
    -3
    -1
     0
```

$$\begin{aligned} H(s) &= -\frac{0.1667}{s-(-3)} - \frac{0.5000}{s-(-1)} + \frac{0.6667}{s-0} \\ &= -\frac{1/6}{s+3} - \frac{1/2}{s+1} + \frac{2/3}{s} \end{aligned}$$

Distinct Complex Roots

Partial fraction expansion applies as well to distinct complex roots. Note that if root r_1 is complex, then the complex conjugate r_1^* is also a root. It can also be shown that the residue c_2 corresponding to the root r_2 is equal to the conjugate c_1^* of the residue corresponding to the root r_1.

Example:

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

```
>> b = [1 -2 1];
>> a = [1 3 4 2];
>> [c,r] = residue(b,a)
c =
  -1.5000+ 2.0000i
  -1.5000- 2.0000i
   4.0000
r =
  -1.0000+ 1.0000i
  -1.0000- 1.0000i
  -1.0000
```

$$H(s) = \frac{-1.5 + j2}{s + 1 - j} + \frac{-1.5 - j2}{s + 1 + j} + \frac{4}{s+1}$$

Repeated Roots

Again consider the general case where $m < n$ and

$$H(s) = \frac{B(s)}{A(s)}$$

Suppose that root r_1 of $A(s)$ is repeated p times and the other $n-p$ roots (denoted $r_{p+1}, r_{p+2}, \ldots, r_n$) are distinct. Then $H(s)$ has the partial fraction expansion

$$H(s) = \frac{c_1}{s - r_1} + \frac{c_2}{(s - r_1)^2} + \cdots + \frac{c_p}{(s - r_1)^p} + \frac{c_{p+1}}{s - r_{p+1}} + \cdots + \frac{c_n}{s - r_n}$$

Example:

$$H(s) = \frac{5s - 1}{s^3 - 3s - 2}$$

```
>> b = [5 -1];
>> a = [1 0 -3 -2];
>> [c,r] = residue(b,a)
c =
    1.0000
   -1.0000
    2.0000
r =
    2.0000
   -1.0000
   -1.0000
```

$$H(s) = \frac{1}{s - 2} - \frac{1}{s + 1} + \frac{2}{(s + 1)^2}$$

Expansion of Improper Rational Functions

Again consider the rational function

$$H(s) = \frac{B(s)}{A(s)}$$

with the degree of $B(s)$ greater than or equal to the degree of $A(s)$ ($m \geq n$). By polynomial division, $H(s)$ can be written in the form

$$H(s) = Q(s) + \frac{R(s)}{A(s)}$$

where the quotient $Q(s)$ is a polynomial in s with degree $m - n$, and the remainder $R(s)$ is a polynomial in s with degree strictly less than n. Thus, $R(s)/A(s)$ is a proper rational function that can be expanded using partial fraction expansion.

The expansion of an improper rational function in MATLAB is computed with a variation of the `residue` function:

`[c,r,q] = residue(b,a)` finds the residues, roots and quotient of a partial fraction expansion of the ratio of two polynomials B(s)/A(s). The coefficients of the quotient Q(s) are returned in the row vector `q`. The number of roots is `n = length(a)-1 = length(c) = length(r)`. The quotient coefficient vector is empty if `length(b) < length(a)`, otherwise `length(q) = length(b)-length(a)+1`.

Example:

$$H(s) = \frac{s^3 + 2s - 4}{s^2 + 4s - 2}$$

```
>> b = [1 0 2 -4];
>> a = [1 4 -2];
>> [c,r,q] = residue(b,a)
c =
   20.6145
   -0.6145
r =
   -4.4495
    0.4495
q =
     1    -4
```

$$H(s) = s - 4 + \frac{20.6145}{s + 4.4495} - \frac{0.6145}{s - 0.4495}$$

Recovering the Rational Function

The coefficients of the rational function can be recovered from the residues, the roots, and the coefficients of the quotient term using yet another form of the `residue` function.

`[b,a] = residue(c,r,q)`, with three input arguments and two output arguments, converts the partial fraction expansion back to the polynomials with coefficients in `b` and `a`.

Consider the continuation of the example above.

```
>> [b,a] = residue(c,r,q)
b =
    1.0000         0    2.0000   -4.0000
a =
     1     4    -2
```

$$H(s) = \frac{s^2 + 2s - 4}{s^2 + 4s - 2}$$

6.4 Functions of Two Variables

To evaluate a function of two variables $f(x, y)$, first define a **two-dimensional grid** in the xy plane, then evaluate the function at the grid points to determine points on the **three-dimensional surface**. This is shown in Figure 6.10, where the surface values $z = f(x, y)$ are plotted above the grid of xy values.

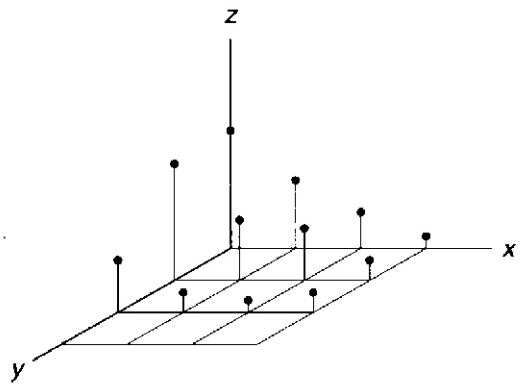

Figure 6.10: Function of two variables plotted in three dimensions

A two-dimensional grid in the xy plane is defined in MATLAB by two vectors, one containing the x-coordinates at all the points in the grid, and the other containing the y-coordinates. For example, to define a grid in x varying from -2 to 2 in increments of 1 and a grid in y varying from -1 to 2 in increments of 1, using colon notation:

```
>> x = -2:2
x =
    -2    -1     0     1     2
>> y = -1:2
y =
    -1     0     1     2
```

The `meshgrid` function generates two matrices that define the underlying grid for a two-dimensional function.

`[X,Y] = meshgrid(x,y)`	Transforms the domain specified by vectors `x` and `y` into arrays `X` and `Y` that can be used for the evaluation of functions of two variables and 3-D surface plots. The rows of the output array `X` are copies of the vector `x` and the columns of the output array `Y` are copies of the vector `y`. If `x` has length n and `y` has length m, then `X` and `Y` are $m \times n$ arrays.
`[X,Y] = meshgrid(x)`	Abbreviation for `[X,Y] = meshgrid(x,x)`

For example:

```
>> [X,Y] = meshgrid(x,y)
```

```
X =
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
    -2    -1     0     1     2
Y =
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
     2     2     2     2     2
```

After the underlying grid matrices have been defined, the corresponding function values can be computed. For example, for the following function

$$f(x, y) = ye^{-(x^2+y^2)}$$

the function values would be computed as

```
Z = Y.* exp(-(X.^2 + Y.^2));
```

The operations are element-by-element, so the value `Z(1,1)` is computed from `X(1,1)` and `Y(1,1)`, and so on. Note that `Z` must be computed from `X` and `Y` and not from `x` and `y`, a common error.

Three-dimensional plots

Among several ways to plot a three-dimensional (3D) surface in MATLAB, a **mesh plot** and a **surface plot** will be described, followed by a description of a **contour plot**. For further information on 3D plots, type `help graph3d`.

`mesh(x_pts,y_pts,Z)`	Generates an open mesh plot of the surface defined by matrix `Z`. The arguments `x_pts` and `y_pts` can be vectors defining the ranges of the values of the x- and y-coordinates, or they can be matrices defining the underlying grid of x- and y-coordinates.
`surf(x_pts,y_pts,Z)`	Generates a shaded mesh plot of the surface defined by matrix `Z`. The arguments `x_pts` and `y_pts` can be vectors defining the ranges of the values of the x- and y-coordinates, or they can be matrices defining the underlying grid of x- and y-coordinates.

For example, to plot the function $f(x, y)$ in the example above, the following commands are executed

```
% Mesh and surface plots of a function of two variables
%
x = -2:0.1:2;
y = -1:0.1:2;
```

```
[X Y] = meshgrid(x,y);
Z = Y.*exp(-(X.^2 + Y.^2));
subplot(2,1,1),mesh(X,Y,Z),...
  title('Mesh Plot'),xlabel('x'),...
  ylabel('y'),zlabel('z'),...
subplot(2,1,2),surf(X,Y,Z),...
  title('Surface Plot'),xlabel('x'),...
  ylabel('y'),zlabel('z')
```

Note that the xy grid has been made finer by incrementing both x and y by 0.1. Also note that the arguments `X` and `Y` could have been replaced by `x` and `y` in both the `mesh` and `surf` commands. The resulting plots are shown in Figure 6.11. Note that you need to know something about the

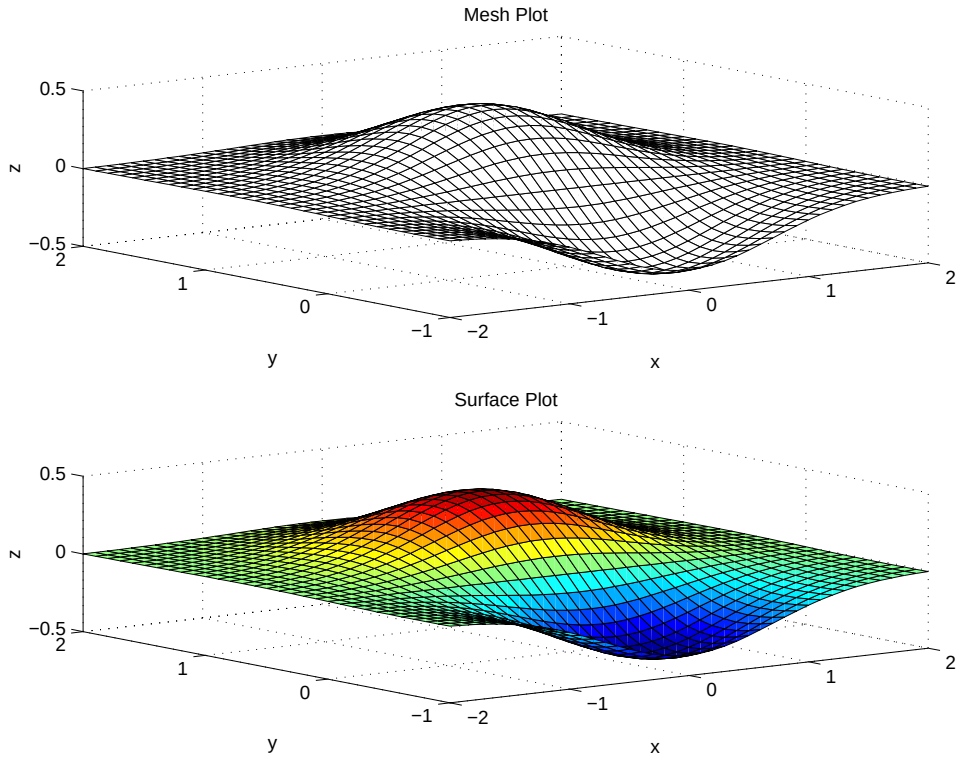

Figure 6.11: Mesh and surface plots of a function of two variables

properties of the two-dimensional function $f(x, y)$ to know what range of values on x and y that you want it to be plotted.

A **contour plot** is an elevation or topographic map consisting of curves representing equal elevations or values of z, called contours of constant elevation.

`contour(x,y,Z)`	Generates a contour plot of the surface defined by the matrix `Z`. The arguments `x` and `y` are vectors defining the ranges of values of the x- and y-coordinates. The number of contour lines and their values are chosen automatically.
`contour(x,y,Z,v)`	Generates a contour plot of the surface defined by the matrix `Z`. The arguments `x` and `y` are vectors defining the ranges of values of the x- and y-coordinates. The vector `v` defines the values to use for the contour lines.
`meshc(x_pts,y_pts,Z)`	Generates an open mesh plot of the surface defined by the matrix `Z`. The arguments `x_pts` and `y_pts` can be vectors defining the ranges of values of the x- and y-coordinates or they can be matrices defining the underlying grid of x- and y-coordinates. In addition, a contour plot is generated below the mesh plot.

The commands to produce the contour plot shown in Figure 6.12 from the function $f(x, y)$ considered in the examples above:

```
contour(x,y,Z),...
  title('Contour Plot'),xlabel('x'),...
  ylabel('y'),grid
```

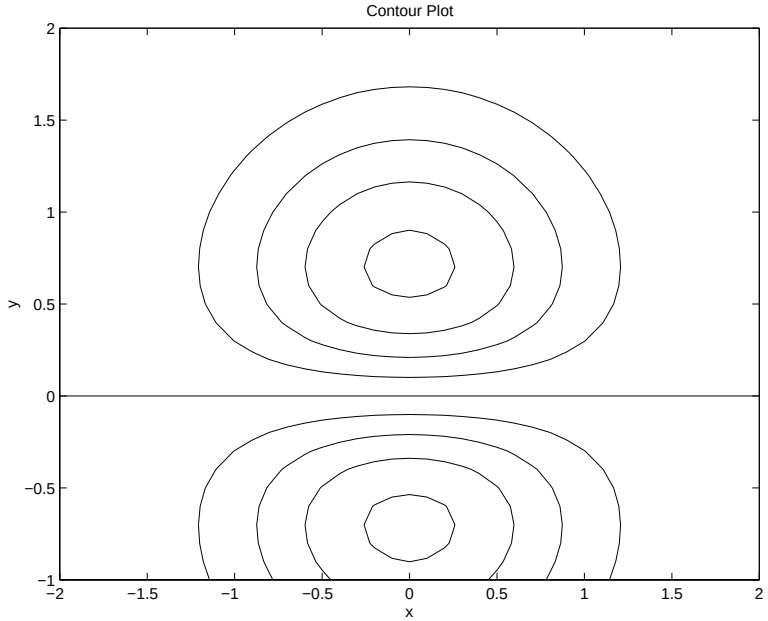

Figure 6.12: Contour plot of a function of two variables

The commands to produce the mesh/contour plot shown in Figure 6.13 from the function $f(x, y)$ considered in the examples above:

```
meshc(X,Y,Z),...
```

```
title('Mesh/Contour Plot'),xlabel('x'),...
ylabel('y'),zlabel('z')
```

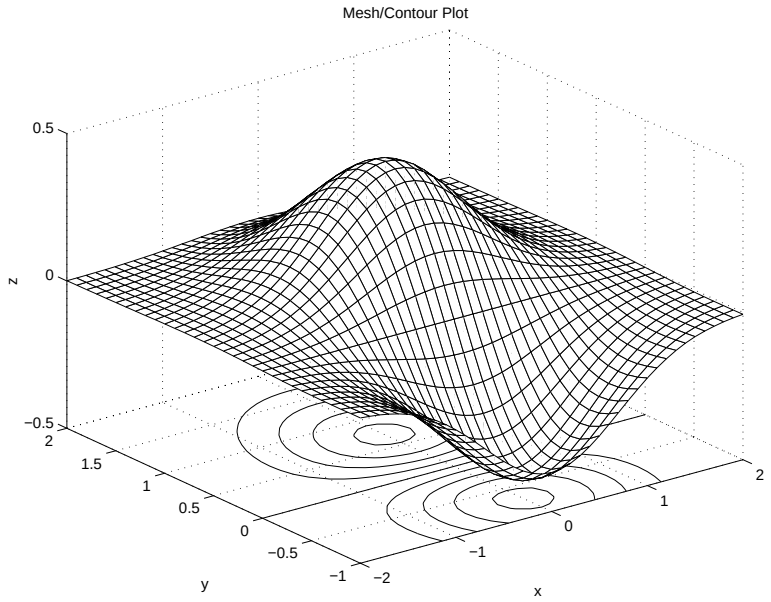

Figure 6.13: Mesh/contour plot of a function of two variables

6.5 User-Defined Functions

For more information, type `help function` in MATLAB.

If you find that you are often building a function from several MATLAB commands, you can develop a **user-defined function** that can can be used in a same way as the built-in MATLAB functions.

A user-defined function is similar to a script file in that it is a text file having a `.m` extension and it is thus called a function M-file. Their variables are *local*, meaning that their values are available only within the function. They are the building blocks of larger scripts, facilitating a modular approach to the development of scripts.

For example, consider writing a function to compute the sine of an angle, with the angle in degrees rather than radians. The user-defined function is the following:

```
function y = sind(x)
%  SIND   Sine in degrees
%    SIND(X) is the sine of the elements of X, in degrees
y = sin(x*pi/180);
```

This function is written to a file named `sind.m`. MATLAB programs and scripts can refer to this function in the same way that they refer to functions such as `sqrt` and `abs`. An example of the use of this to simplify a command in the script written for Example 4.2 is:

```
>> AC = 245/sind(30)
AC =
  490.0000
```

The rules for writing an M-file function are the following:

1. **Function definition line:** The first line of a function has the following syntax:

   ```
   function [output variables] = function_name(input variables);
   ```

   Thus, it must contain the word `function`, followed by the output variables, an equal sign, and the name of the function. The input variables, called arguments, of the function follow the function name and are enclosed in parentheses. This line distinguishes the function file from a script file. The definition line for the example above is:

   ```
   function y = sind(x)
   ```

   The output variable is `y`, the function name is `sind`, and the input argument is `x`.

2. **Function call:** A user-defined function is called by the name of the M-file in which it is defined, *not* by the name given the function in the first line of the file. Thus, if the function script above were renamed `dsin.m`, but the script itself were unchanged, then it would have to be called by the name `dsin`, as follows:

   ```
   >> y = dsin(30)
   y =
       0.5000
   ```

   An attempt to call it by the function name `sind` results in an error message:

   ```
   >> y = sind(30)
   ??? Undefined function or variable 'sind'.
   ```

   To avoid confusion, use the same name for the function and the M-file.

3. **Comments:** The first few lines should be comments, as they will be displayed if `help` is requested for the function name. The first comment line is referenced by the `lookfor` command. Each comment line must start with a percent character (%). For the example above:

   ```
   >> help sind

      SIND   Sine in degrees
        SIND(X) is the sine of the elements of X, in degrees
   ```

4. **Information returned:** The only information returned from the function is contained in the output variables (also called output arguments). A statement must always be included that assigns a value to the output variables specified in the function definition line. These output variables will be arrays. Thus, while we thought of the function `sind` as operating on a scalar and returning a scalar, it can also operate on an array and return an array:

```
>> lengths = 100*sind([30 60 90; 120 150 180])
lengths =
   50.0000   86.6025  100.0000
   86.6025   50.0000    0.0000
```

Note that output variables are optional. This allows a function to be written to perform an operation such as toggling `diary`, but not to return any information.

5. **Communication:** A function communicates with the MATLAB workspace only through the variables passed to it and through the output variables it creates. Intermediate variables within the function do not appear in, or interact with, the MATLAB workspace. Thus, each function has its own workspace separate from the MATLAB workspace. Variables in the function M-file that are not output are input variables are said to be *local* to the function.

6. **Multiple outputs:** To return more than one output variable, they must be listed in a vector, separated by commas, as in the following example that will return three variables:

   ```
   function [distance, velocity, accel] = position(x)
   ```

   Since the function is returning multiple arguments, it must be used as follows:

   ```
   >> [d v a] = position(A)
   ```

7. **Multiple inputs:** When there are multiple input arguments, they must be separated by commas. For example:

   ```
   function c = hypot(a,b)
   ```

8. **Semicolons:** The use of semicolons (;) at the end of commands in a function script have the same purpose that they serve an any other MATLAB command, suppressing display of the results of the command. In most cases, it is not desirable to display the results of commands internal to a function.

Many of the commands available in MATLAB are written as function M-files. You can find the locations of these files using the `which` command. For example:

```
>> which linspace
/usr/pkg/matlab52/toolbox/matlab/elmat/linspace.m
```

You could display this M-file using the `type` command and take ideas from the function script for use in writing your own function scripts.

Application: Minimizing a Function of One Variable

To find the minimum of a function of a single variable:

`x = fminbnd('F',x1,x2)`	Returns a value of `x` that is the local minimizer of `F(x)` in the interval [`x1`, `x2`], where `F` is a string containing the name of the function.
`[x,fval] = fminbnd(...)`	Also returns the value of the objective function, `fval`, computed in `F`, at `x`.

For example:

```
>> fminbnd('cos',0,4)
ans =
    3.1416
```

To use this command to find the minimum of more complicated functions, it is convenient to define the function in a function M-file. For example, consider the polynomial function

$$y = 0.025x^5 - 0.0625x^4 - 0.333x^3 + x^2$$

Defining the function file `fp5.m`:

```
function y = fp5(x)
% FP5, fifth degree polynomial function
y = 0.025*x.^5 - 0.0625*x.^4 - 0.333*x.^3 + x.^2
```

Observe in Figure 6.5 that this function has two minima in the interval $-1 \leq x \leq 4$. The minimum near $x = 3$ is called a *relative* or *local* minimum because it forms a valley whose lowest point is higher than the minimum at $x = 0$. The minimum at $x = 0$ is the true minimum and is also called the *global* minimum. Searching for the minimum over the interval $-1 \leq x \leq 4$:

```
>> xmin = fminbnd('fp5',-1,4)
xmin =
  2.0438e-006
```

The resulting value for `xmin` is essentially 0, the true minimum point. Searching for the minimum over the interval $0.1 \leq x \leq 2.5$:

```
>> xmin = fminbnd('fp5',0.1,2.5)
xmin =
    0.1001
```

This misses the true minimum point, as it is not included in the specified interval. Also, `fminbnd` can give incorrect answers. If the interval is $1 \leq x \leq 4$:

```
>> xmin = fminbnd('fp5',1,4)
xmin =
    2.8236
```

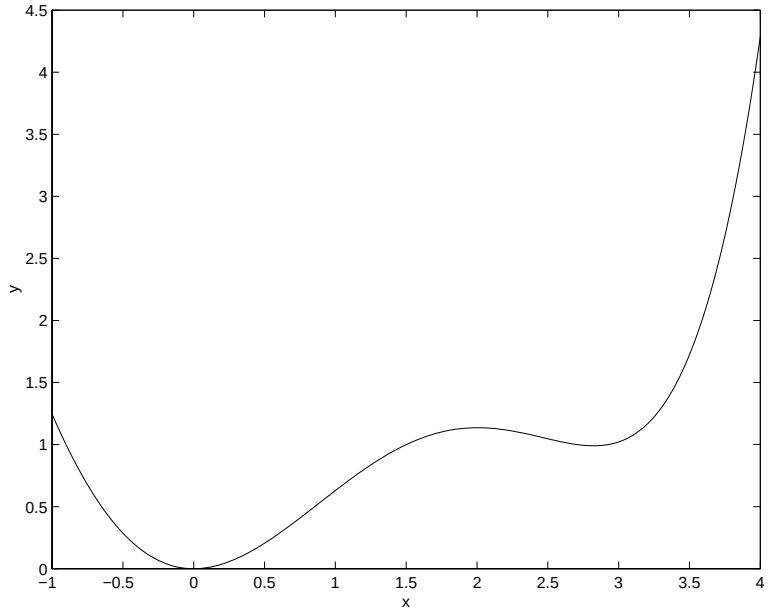

Figure 6.14: Plot of the function $y = 0.025x^5 - 0.0625x^4 - 0.333x^3 + x^2$

The result corresponds to the "valley" shown in the plot, but which is not the minimum point on this interval, which is at the boundary $x = 1$. The `fminbnd` function first looks for a minimum point corresponding to a zero slope; if it finds one, it stops. If it does not find one, it looks at the function values at the boundaries of the specified interval for x. In this example, a zero-slope minimum was found, so the true minimum at the boundary was missed.

6.6 Plotting Functions

As described previously, a function such as a polynomial can be evaluated and then plotted with the `plot` command. This can also be done in one step with the `fplot` command.

`fplot(fun,lims)` plots the function specified by the string `fun` between the x-axis limits specified by `lims = [xmin xmax]`. Using `lims = [xmin xmax ymin ymax]` also controls the y-axis limits. `fun` must be the name of an M-file function or a string with variable `x`, such as `sin(x)`, `diric(x,10)` or `[sin(x),cos(x)]`. The function `fun(x)` must return a row vector for each element of vector `x`.

Consider the plotting the following function, known as the *sinc* or *sampling* function:

$$\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

This is a function available in MATLAB, which can be plotted with the command:

```
fplot('sinc',[-10 10]),ylabel('sinc(x)'),xlabel('x')
```

This generates the plot shown in Figure 6.15

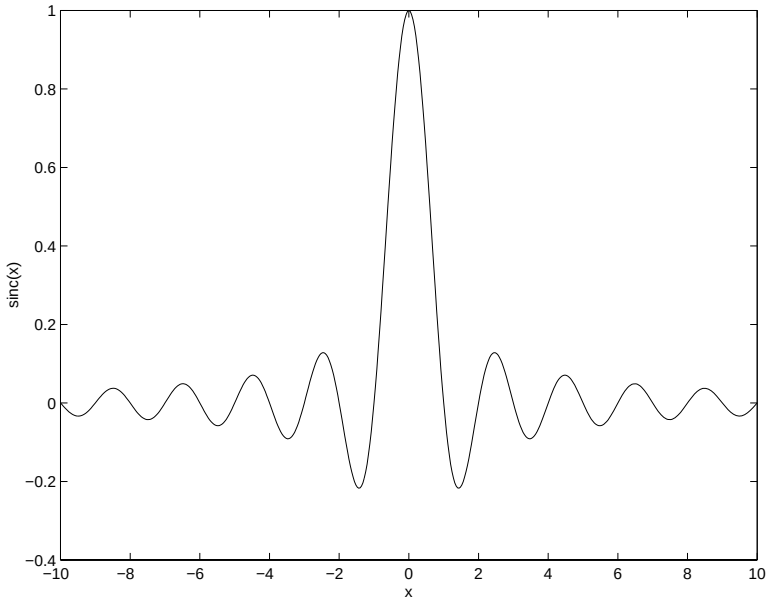

Figure 6.15: Sinc signal