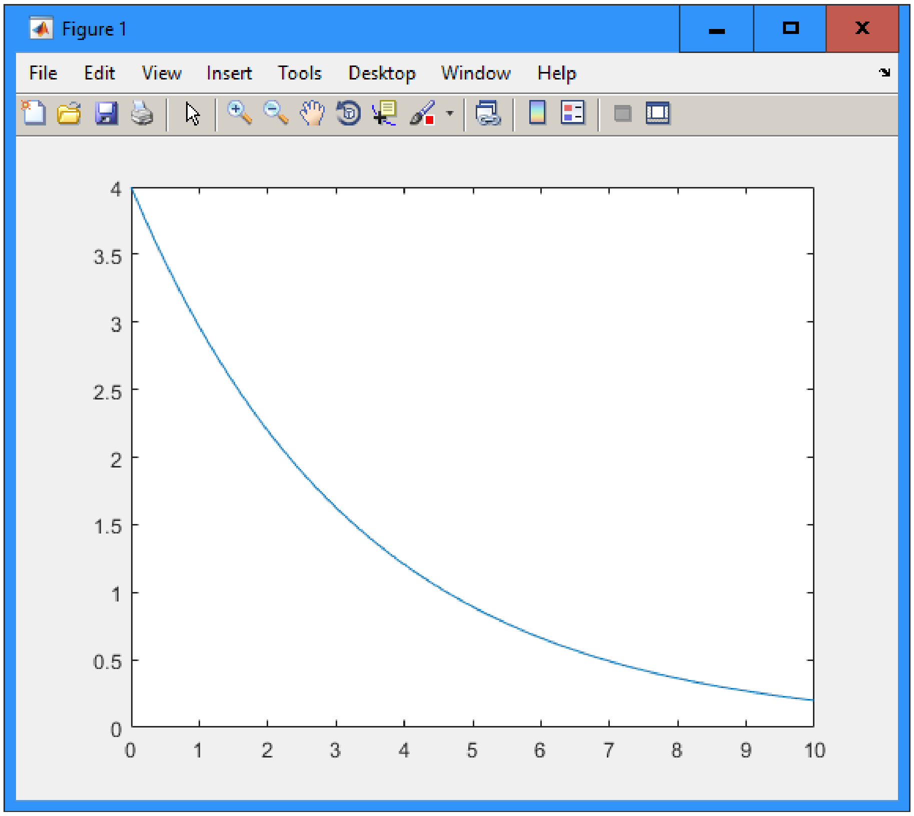
Table of Contents

1. Introduction to MATLAB

1.1 When these statements are executed, the results are as shown below:



Exercises 1.2 through 1.3 are procedural exercises, and do not appear in this Solutions Manual.

1.4 *(a)*

```
>> (1/(5^2) + 3/2*pi - 1)^(-3)
ans =
    0.0189
```

(b)

```
» 2*pi - pi^0.5
ans =
    4.5107
```

(c)

```
» 1 + ½ + 1/2^2 + 1/3^3 + 1/2^4
ans =
    1.8495
```

1.5 A MATLAB Command Window session that evaluates the specified expressions is shown below. In this and all future exercises, user inputs are shown in bold face.

```
>> u=1;
>> v=3;
>> (4*u) / (3*v)
ans =
    0.4444
>> (2*v^-2) / (u+v)^2
ans =
    0.0139
>> v^3/(v^3 - u^3)
ans =
    1.0385
>> (4/3)*pi*v^2
ans =
   37.6991
>> u*sqrt(v) + 1
ans =
    2.7321
>> log10( (v+u) / (v-u) )
ans =
    0.3010
```

1.6 A script file that executes all of the equations from Exercise 1.5 is given below.

```
u = 1;
v = 3;

(4*u) / (3*v)
(2*v^-2) / (u+v)^2
v^3/(v^3 - u^3)
(4/3)*pi*v^2
u*sqrt(v) + 1
log10( (v+u) / (v-u) )
```

When this script file is executed, the results are:.

```
>> test
ans =
    0.4444
ans =
    0.0139
ans =
    1.0385
ans =
   37.6991
ans =
    2.7321
ans =
    0.3010
```

1.7 Suppose that x = 2 and y = -1. Then

(a)

```
>> (2 * x^3) ^ (1/4)
ans =
     2
```

(b)

```
» (2 * y^3) ^ (1/4)
ans =
   0.8409 + 0.8409i
```

1.8 The area inside an ellipse with $a = 5$ and $b = 10$ is given by:

```
>> a = 5;
>> b = 10;
>> pi*a*b
ans =
  157.0796
```

1.9 A script file to calculate the approximate circumference of an ellipse with axes $a = 5$ and $b = 10$ is given below:

```
% Calculate the circumference of an ellipse with axes
% a = 5 and b = 10

a = 5
b = 10

h = (a-b).^2 / (a+b).^2

circ = pi * (a+b) * (1+ (3*h)/(10+sqrt(4-3*h)))
```

When this script file is executed, the result is:

```
>> calc_circumference
a =
     5
b =
    10
h =
    0.1111
circ =
   48.4422
```

1.10 The script file `circle_and_shpere2.m` is shown below:

```
% Calculate the area and circumference of a circle
% if radius r, and the volume and surface area of a
% sphere of radius r
area_circle = pi * r^2
circumference_circle =  2 * pi * r
volume_sphere = 4 / 3 * pi * r^3
```

```
area_sphere = 4 * pi * r^2
```

This script can be used to calculate the circle and sphere parameters for radii 1, 5, 10, and 20 as follows:

```
>> r = 1
r =
     1
>> circle_and_sphere2
area_circle =
    3.1416
circumference_circle =
    6.2832
volume_sphere =
    4.1888
area_sphere =
   12.5664
>> r = 5
r =
     5
>> circle_and_sphere2
area_circle =
   78.5398
circumference_circle =
   31.4159
volume_sphere =
  523.5988
area_sphere =
  314.1593
>> r = 10
r =
    10
>> circle_and_sphere2
area_circle =
  314.1593
circumference_circle =
   62.8319
volume_sphere =
   4.1888e+03
area_sphere =
   1.2566e+03
>> r = 20
r =
    20
>> circle_and_sphere2
area_circle =
   1.2566e+03
circumference_circle =
  125.6637
volume_sphere =
   3.3510e+04
area_sphere =
   5.0265e+03
```

1.11 The values are a = 62.8319, b = 6.3662, and ans = 20. After the line '4 * 5' is executed, the variable `ans` contains 20. When the next two lines are executed, the value 20 stored in `ans` is used in the calculations. Since those calculations are assigned to variables `a` and `b` respectively, the value in `ans` is not changed by those lines. Therefore, it is still 20 when it is printed out at the end.

1.12 The required command to determine the current directory is `pwd`. The default directory when my installation of MATLAB starts is.

```
>> pwd
ans =
C:\Users\schapman\Documents\MATLAB
```

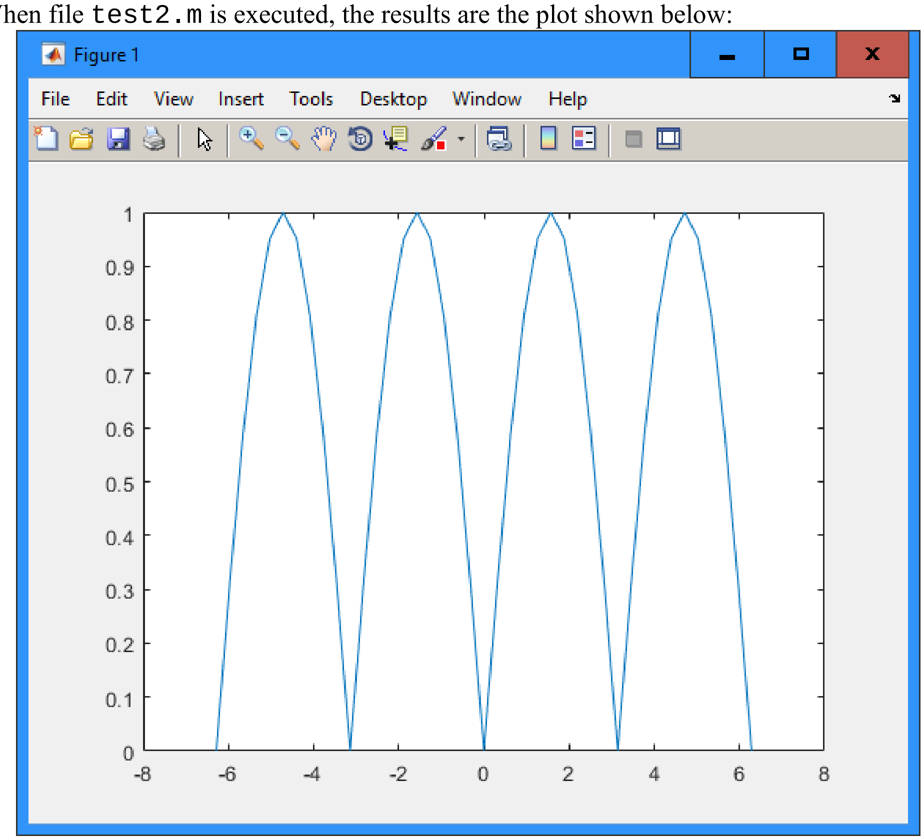
The current directory on startup will be different on your computer.

1.13 The required command to create a new directory is `mkdir`. The command to add the directory to the path is `addpath`. The directory can be created and added to the path with the statements.

```
mkdir('mynewdir');
addpath('mynewdir');
```

Alternately, both jobs can be performed using the Path Tool (`pathtool`).

1.14 When file `test2.m` is executed, the results are the plot shown below:



1.15 When `test2` is typed into the Command Window, file `test2.m` is executed, even though it is not in the current directory, because it is in the MATLAB search path. The results are identical to before.

2. MATLAB Basics

2.1 *(a)* The size of `array1` is 4 × 5. *(b)* The value of `array1(1,4)` is -3.5. *(c)* The value of `array1(9)` is 2.1. *(d)* `array1(:,1:2:4)` is a 4 × 2 array consisting of the first and third columns of `array1`:

```
>> array1(:,1:2:4)
ans =
         0    2.1000
   -0.1000   -6.6000
    1.2000    0.5000
    1.1000         0
```

(e) `array1([1 3],[end-1 end])` consists of the elements in the first and third rows on the last two columns of `array1`:

```
» array1([1 3],end)
ans =
   -3.5000    5.0000
   -0.4000    1.3000
```

2.2 *(a)* Legal *(b)* Illegal—names must begin with a letter. *(c)* Illegal—ampersands are illegal characters *(d)* Legal *(e)* Illegal —names must begin with a letter. *(f)* Illegal—the apostrophe and the question mark are illegal characters.

2.3 *(a)* This is a four element row array containing the values 2, 5, 8, and 11:

```
» a = 2:3:12
a =
     2     5     8    11
```

(b) This is a 4 × 3 element array containing three identical columns:

```
» b = [a' a' a']
b =
     2     2     2
     5     5     5
     8     8     8
    11    11    11
```

(c) This is a 2 × 2 element array containing the first and third rows of the first and third columns of `b`:

```
» c = b(1:2:3,1:2:3)
c =
     2     2
     8     8
```

(d) This is a 1 × 3 row array containing the sum of `a(2:4)` (= [5 8 11]) plus the second row of `b` (= [5 5 5]):

```
» d = a(2:4) + b(2,:)
d =
    10    13    16
```

(e) This is a 1 × 9 row array containing:

```
» w = [zeros(1,3) ones(3,1)' 3:5']
w =
     0     0     0     1     1     1     3     4     5
```

Note that the expression `3:5'` is the same as `3:5`, because the transpose operator applies to the single element 5 only: `5' = 5`. Both expressions produce the row array `[1 3 5]`. To produce a column array, we would write the expression as `(3:5)'`, so that the transpose operator applied to the entire vector.

(f) This statement swaps *the first and third rows in the second column* of array `b`:

```
» b([1 3],2) = b([3 1],2)
b =
     2     8     2
     5     5     5
     8     2     8
    11    11    11
```

(g) This statement produces nothing, because even the first element (1) is below the termination condition (5) when counting down:

```
» e = 1:-1:5
e =
   Empty matrix: 1-by-0
```

2.4 *(a)* This is the fourth row of the array:

```
» array1(4,:)
ans =
   -1.4000    7.2000   -2.6000    1.1000   -3.0000
```

(b) This is the fourth column of the array:

```
» array1(:,4)
ans =
   -3.5000
    2.8000
   -0.4000
    1.1000
```

(c) This array consists of the first and third rows and the third and fourth columns of `array1`, with the third column *repeated twice*:

```
» array1(1:2:3,[3 3 4])
ans =
   -2.1000   -2.1000   -3.5000
```

```
    0.1000    0.1000   -0.4000
```

(d) This array consists of the third row *repeated twice*:

```
» array1([3 3],:)
ans =
    2.1000    0.5000    0.1000   -0.4000    5.3000
    2.1000    0.5000    0.1000   -0.4000    5.3000
```

2.5 *(a)* This statement displays the number using the normal MATLAB format:

```
» disp (['value = ' num2str(value)]);
value = 31.4159
```

(b) This statement displays the number as an integer:

```
» disp (['value = ' int2str(value)]);
value = 31
```

(c) This statement displays the number in exponential format:

```
» fprintf('value = %e\n',value);
value = 3.141593e+01
```

(d) This statement displays the number in floating-point format:

```
» fprintf('value = %f\n',value);
value = 31.415927
```

(e) This statement displays the number in general format, which uses an exponential form if the number is too large or too small.

```
» fprintf('value = %g\n',value);
value = 31.4159
```

(f) This statement displays the number in floating-point format in a 12-character field, with 4 digits after the decimal point:

```
» fprintf('value = %12.4f\n',value);
value =      31.4159
```

2.6 The results of each case are shown below.

(a) Legal: This is element-by-element addition.

```
» result = a + b
result =
     1     4
    -1     6
```

(b) Legal: This is matrix multiplication. Since `eye(2)` is the 2×2 identity matrix $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, the result of the multiplication is just the original matrix `a`.

```
» result = a * d
result =
     2     1
    -1     4
```

(c) Legal: This is element by element array multiplication

```
» result = a .* d
result =
     2     0
     0     4
```

(d) Legal: This is matrix multiplication

```
» result = a * c
result =
     5
     2
```

(e) Legal: This is element by element array multiplication. Since array `c` has only one column, the column dimension is 1, and it is copied to make the array be the same size as `a` ($\begin{bmatrix} 2 & 2 \\ 1 & 1 \end{bmatrix}$) before performing the multiplication.

```
» result = a .* c
result =
     4     2
    -1     4
```

(f) Legal: This is matrix left division

```
» result = a \ b
result =
   -0.4444    1.1111
   -0.1111    0.7778
```

(g) Legal: This is element by element array left division: `b(i) / a(i)`

```
» result = a .\ b
result =
   -0.5000    3.0000
         0    0.5000
```

(h) Legal: This is element by element exponentiation

```
» result = a .^ b
result =
    0.5000    1.0000
    1.0000   16.0000
```

2.7 *(a)* 6.4 *(b)* 6.4 *(c)* 1.3333 *(d)* 729 *(e)* 6561 *(f)* 729 *(g)* 2 *(h)* 2 *(i)* 1

2.8 *(a)* 0.0 + 25.0*i* *(b)* -0.6224*i*

2.9 *(a)* 0.0183 *(b)* 0.0183 - 0.0000*i* *(c)* 0.0183 + 0.0000*i*. Note that the answers to *(b)* and *(c)* are identical, because they are the two equivalent forms of Euler's Equation.

2.10 The solution to this set of equations can be found using the left division operator:

```
» a = [ -2.0 +5.0 +2.0 +3.0 +4.0 -1.0; ...
         2.0 -1.0 -5.0 -2.0 +6.0 +4.0; ...
        -1.0 +6.0 -4.0 -5.0 +3.0 -1.0; ...
         4.0 +3.0 -6.0 -5.0 -2.0 -2.0; ...
        -3.0 +6.0 +4.0 +2.0 -5.0 +4.0; ...
         2.0 +4.0 +4.0 +4.0 +5.0 -4.0 ];

» b = [ -3.0; 1.0; -6.0; 10.0; -6.0; -2.0];
» a\b
ans =
    1.1021
   -0.2385
   -2.1454
    1.9457
   -0.9754
   -0.6064
```

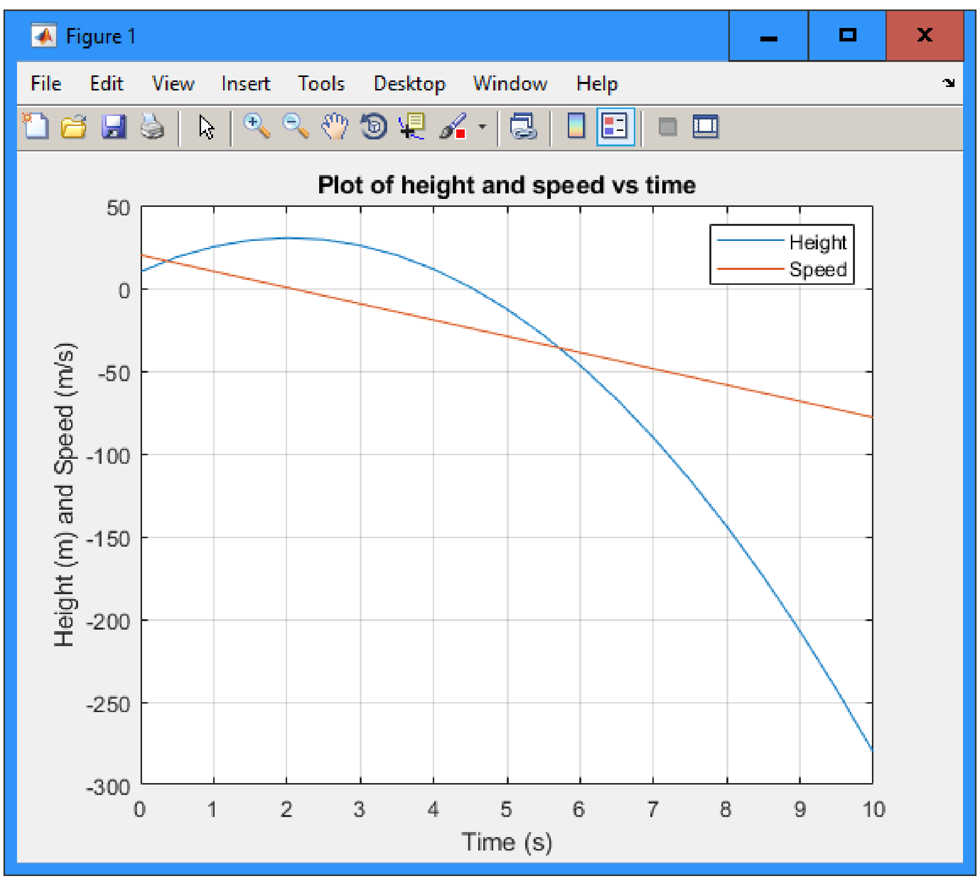
2.11 A program to plot the height and speed of a ball thrown vertically upward is shown below:

```
%  Script file: ball.m
%
%  Purpose:
%    To calculate and display the trajectory of a ball
%    thrown upward at a user-specified height and speed.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   g         -- Acceleration due to gravity (m/s^2)
%   h         -- Height (m)
%   h0        -- Initial height (m)
%   t         -- Time (s)
%   v         -- Vertical Speed (m/s)
%   v0        -- Initial Vertical Speed (m/s)

% Initialize the acceleration due to gravity
g = -9.81;

% Prompt the user for the initial velocity.
v0 = input('Enter the initial velocity of the ball: ');

% Prompt the user for the initial height
h0 = input('Enter the initial height of the ball: ');
```

```
% We will calculate the speed and height for the first
% 10 seconds of flight.  (Note that this program can be
% refined further once we learn how to use loops in a
% later chapter.  For now, we don't know how to detect
% the point where the ball passes through the ground
% at height = 0.)
t = 0:0.5:10;
h = zeros(size(t));
v = zeros(size(t));
h = 0.5 * g * t .^2 + v0 .* t + h0;
v = g .* t + v0;

% Display the result
plot(t,h,t,v);
title('Plot of height and speed vs time');
xlabel('Time (s)');
ylabel('Height (m) and Speed (m/s)');
legend('Height','Speed');
grid on;
```

When this program is executed, the results are:

```
» ball
Enter the initial velocity of the ball: 20
Enter the initial height of the ball: 10
```



2.12 A program to calculate the distance between two points in a Cartesian plane is shown below:

```
%  Script file: dist2d.m
%
%  Purpose:
%    To calculate the distance between two points on a
%    Cartesian plane.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   dist      -- Distance between points
%   x1, y1    -- Point 1
%   x2, y2    -- Point 2

% Prompt the user for the input points
x1 = input('Enter x1: ');
y1 = input('Enter y1: ');
x2 = input('Enter x2: ');
y2 = input('Enter y2: ');

% Calculate distance
dist = sqrt((x1-x2)^2 + (y1-y2)^2);

% Tell user
disp (['The distance is ' num2str(dist)]);
```

When this program is executed, the results are:

```
» dist2d
Enter x1: -3
Enter y1: 2
Enter x2: 3
Enter y2: -6
The distance is 10
```

2.13 *(a)* A program that accepts a 2D vector in rectangular coordinates and calculates the vector in polar coordinates is given below;

```
%  Script file: rect2polar.m
%
%  Purpose:
%    To calculate the polar coordinates of a vector given the
%    rectangular coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
```

```
% Define variables:
%   x, y      -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector in degrees

% Prompt the user for the input points
x = input('Enter x: ');
y = input('Enter y: ');

% Calculate polar coordinates.  Note that "180/pi" converts
% from radians to degrees.
r = sqrt(x^2 + y^2);
theta = atan2(y,x) * 180/pi;

% Tell user
disp (['The polar coordinates are ' num2str(r) ' at an angle of ' ...
num2str(theta) ' deg.']);
```

When this program is executed, the results are:

```
>> rect2polar
Enter x: 3
Enter y: 4
The polar coordinates are 5 at an angle of 53.1301 deg.
```

(b) A program that accepts a 2D vector in polar coordinates and calculates the vector in rectangular coordinates is given below;

```
%  Script file: polar2rect.m
%
%  Purpose:
%    To calculate the rectangular coordinates of a vector given
%    the polar coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   x, y      -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector in degrees

% Prompt the user for the input points
r = input('Enter vector length r: ');
theta = input('Enter angle theta in degrees: ');

% Calculate polar coordinates.  Note that "pi/180" converts
% from radians to degrees.
x = r * cos(theta * pi/180);
y = r * sin(theta * pi/180);

% Tell user
```

```
disp (['The rectangular coordinates are x = ' num2str(x) ' and y = '  ...
num2str(y)]);
```

When this program is executed, the results are:

```
>> polar2rect
Enter vector length r: 5
Enter angle theta in degrees: 36.87
The rectangular coordinates are x = 3 and y = 4
```

2.14 The difference between these programs and the ones in the previous exercise is that we can drop the degrees to radians conversions. *(a)* A program that accepts a 2D vector in rectangular coordinates and calculates the vector in polar coordinates is given below;

```
%  Script file: rect2polar.m
%
%  Purpose:
%    To calculate the polar coordinates of a vector given the
%    rectangular coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
% 1. 12/21/18    S. J. Chapman        Use atan2d
%
% Define variables:
%   x, y      -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector in degrees

% Prompt the user for the input points
x = input('Enter x: ');
y = input('Enter y: ');

% Calculate polar coordinates.  Note that "180/pi" converts
% from radians to degrees.
r = sqrt(x^2 + y^2);
theta = atan2d(y,x);

% Tell user
disp (['The polar coordinates are ' num2str(r) ' at an angle of ' ...
num2str(theta) ' deg.']);
```

When this program is executed, the results are:

```
>> rect2polar
Enter x: 3
Enter y: 4
The polar coordinates are 5 at an angle of 53.1301 deg.
```

(b) A program that accepts a 2D vector in polar coordinates and calculates the vector in rectangular coordinates is given below;

```
%  Script file: polar2rect.m
%
%  Purpose:
%    To calculate the rectangular coordinates of a vector given
%    the polar coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
% 1. 12/21/18    S. J. Chapman        Use sind and cosd
%
% Define variables:
%   x, y      -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector in degrees

% Prompt the user for the input points
r = input('Enter vector length r: ');
theta = input('Enter angle theta in degrees: ');

% Calculate polar coordinates.  Note that "pi/180" converts
% from radians to degrees.
x = r * cosd(theta);
y = r * sind(theta);

% Tell user
disp (['The rectangular coordinates are x = ' num2str(x) ' and y = '  ...
num2str(y)]);
```

When this program is executed, the results are:

```
>> polar2rect
Enter vector length r: 5
Enter angle theta in degrees: 36.87
The rectangular coordinates are x = 3 and y = 4
```

2.15 A program to calculate the distance between two points in a Cartesian plane is shown below:

```
%  Script file: dist3d.m
%
%  Purpose:
%    To calculate the distance between two points on a
%    three-dimensional Cartesian plane.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   dist        -- Distance between points
%   x1, y1, z1 -- Point 1
%   x2, y2, z2 -- Point 2
```

```
% Prompt the user for the input points
x1 = input('Enter x1: ');
y1 = input('Enter y1: ');
z1 = input('Enter z1: ');
x2 = input('Enter x2: ');
y2 = input('Enter y2: ');
z2 = input('Enter z2: ');

% Calculate distance
dist = sqrt((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2);

% Tell user
disp (['The distance is ' num2str(dist)]);
```

When this program is executed, the results are:

```
» dist3d
Enter x1: -3
Enter y1: 2
Enter z1: 5
Enter x2: 3
Enter y2: -6
Enter z2: -5
The distance is 14.1421
```

2.16 *(a)* A program that accepts a 3D vector in rectangular coordinates and calculates the vector in spherical coordinates is shown below:

```
%  Script file: rect2spherical.m
%
%  Purpose:
%    To calculate the spherical coordinates of a vector given
%    the 3D rectangular coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   x, y, z   -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector (x,y) plane, in degrees
%   phi       -- Elevation angle of vector, in degrees

% Prompt the user for the input points
x = input('Enter x: ');
y = input('Enter y: ');
z = input('Enter z: ');

% Calculate polar coordinates.  Note that "180/pi" converts
% from radians to degrees.
r = sqrt(x^2 + y^2 + z^2);
```

```
theta = atan2(y,x) * 180/pi;
phi = atan2(z,sqrt(x^2 + y^2)) * 180/pi;

% Tell user
disp ('The spherical coordinates are:');
disp (['r     = ' num2str(r)]);
disp (['theta = ' num2str(theta)]);
disp (['phi   = ' num2str(phi)]);
```

When this program is executed, the results are:

```
>> rect2spherical
Enter x: 4
Enter y: 3
Enter z: 0
The spherical coordinates are:
r     = 5
theta = 36.8699
phi   = 0

>> rect2spherical
Enter x: 4
Enter y: 0
Enter z: 3
The spherical coordinates are:
r     = 5
theta = 0
phi   = 36.8699
```

(b) A program that accepts a 3D vector in spherical coordinates (with the angles θ and ϕ in degrees) and calculates the vector in rectangular coordinates is shown below:

```
%  Script file: spherical2rect.m
%
%  Purpose:
%    To calculate the 3D rectangular coordinates of a vector
%    given the spherical coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
%
% Define variables:
%   x, y, z   -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector (x,y) plane, in degrees
%   phi       -- Elevation angle of vector, in degrees

% Prompt the user for the input points
r = input('Enter vector length r: ');
theta = input('Enter plan view angle theta in degrees: ');
phi = input('Enter elevation angle phi in degrees: ');
```

```
% Calculate spherical coordinates.  Note that "pi/180" converts
% from radians to degrees.
x = r * cos(phi * pi/180) * cos(theta * pi/180);
y = r * cos(phi * pi/180) * sin(theta * pi/180);
z = r * sin(phi * pi/180);

% Tell user
disp ('The 3D rectangular coordinates are:');
disp (['x = ' num2str(x)]);
disp (['y = ' num2str(y)]);
disp (['z = ' num2str(z)]);
```

When this program is executed, the results are:

```
>> spherical2rect
Enter vector length r: 5
Enter paln view angle theta in degrees: 36.87
Enter elevation angle phi in degrees: 0
The rectangular coordinates are:
x = 4
y = 3
z = 0
>> spherical2rect
Enter vector length r: 5
Enter paln view angle theta in degrees: 0
Enter elevation angle phi in degrees: 36.87
The rectangular coordinates are:
x = 4
y = 0
z = 3
```

2.17 *(a)* A program that accepts a 3D vector in rectangular coordinates and calculates the vector in spherical coordinates is shown below:

```
%  Script file: rect2spherical.m
%
%  Purpose:
%    To calculate the spherical coordinates of a vector given
%    the 3D rectangular coordinates.
%
%  Record of revisions:
%      Date       Programmer          Description of change
%      ====       ==========          =====================
%    12/21/18    S. J. Chapman        Original code
% 1. 12/21/18    S. J. Chapman        Modified to use cart2sph
%
% Define variables:
%   x, y, z   -- Rectangular coordinates of vector
%   r         -- Length of vector
%   theta     -- Direction of vector (x,y) plane, in degrees
%   phi       -- Elevation angle of vector, in degrees

% Prompt the user for the input points
```