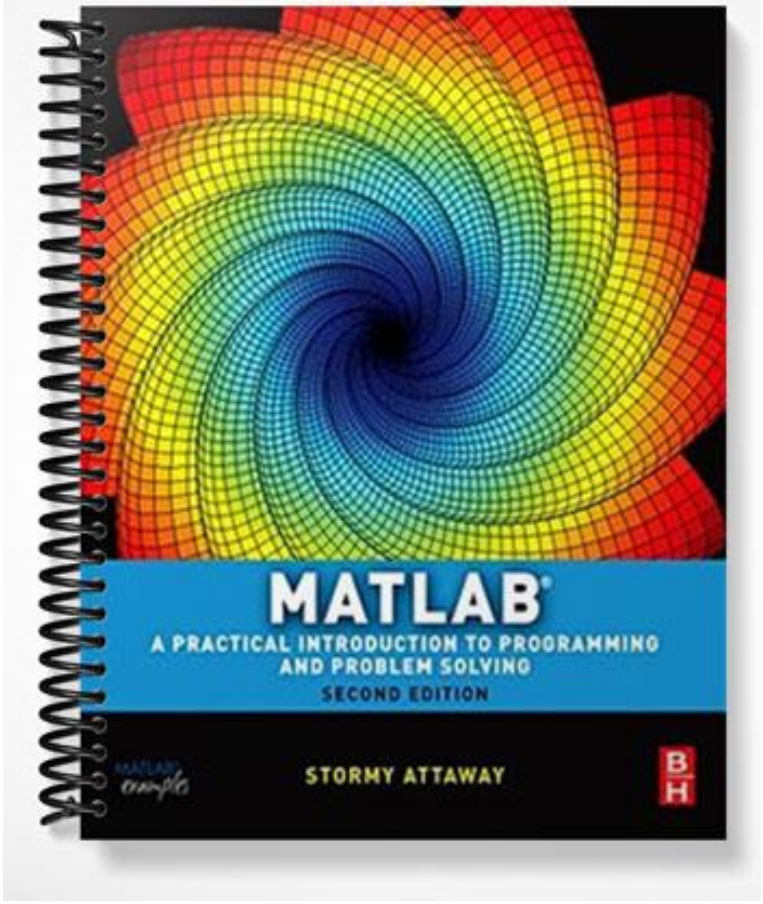
# SOLUTIONS MANUAL



# MATLAB: A Practical Introduction to Programming and Problem Solving

## Second Edition

## SOLUTION MANUAL

Stormy Attaway

College of Engineering
Boston University

# I. Introduction to Programming Using MATLAB

## Chapter 1: Introduction to MATLAB

### Exercises

1) Create a variable to store the atomic weight of silicon (28.085).

```
>> siliconAtWt = 28.085
siliconAtWt =
   28.0850
```

2) Create a variable *myage* and store your age in it. Subtract one from the value of the variable. Add two to the value of the variable.

```
>> myage = 35;
>> myage = myage - 1;
>> myage = myage + 2;
```

3) Use the built-in function **namelengthmax** to find out the maximum number of characters that you can have in an identifier name under your version of MATLAB.

```
>> namelengthmax
ans =
    63
```

4) Explore the **format** command in more detail. Use **help format** to find options. Experiment with **format bank** to display dollar values.

```
>> format +
>> 12.34
ans =
+
>> -123
ans =
-
>> format bank
>> 33.4
ans =
         33.40
>> 52.435
ans =
```

```
          52.44
```

5) Find a **format** option that would result in the following output format:

```
>> 5/16 + 2/7
ans =
     67/112
```

```
>> format rat
>> 5/16 + 2/7
ans =
      67/112
```

6) Think about what the results would be for the following expressions, and then type them in to verify your answers.

```
25 / 4 * 4
3 + 4 ^ 2
4 \ 12 + 4
3 ^ 2
(5 - 2) * 3
```

```
>> 25 / 4 * 4
ans =
      25
```

```
>> 3 + 4 ^ 2
ans =
      19
```

```
>> 4 \ 12 + 4
ans =
       7
```

```
>> 3 ^ 2
ans =
       9
```

```
>> (5 - 2) * 3
ans =
       9
```

7) The combined resistance $R_T$ of three resistors $R_1$, $R_2$, and $R_3$ in parallel is given by

$$R_T = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

Create variables for the three resistors and store values in each, and then calculate the combined resistance.

```
>> r1 = 3;
>> r2 = 2.2;
>> r3 = 1.5;
>> rt = 1/(1/r1 + 1/r2 + 1/r3)
rt =
    0.6875
```

*As the world becomes more "flat", it is increasingly important for engineers and scientists to be able to work with colleagues in other parts of the world. Correct conversion of data from one system of units to another (for example, from the metric system to the American system or vice versa) is critically important.*

8) Create a variable *pounds* to store a weight in pounds. Convert this to kilograms and assign the result to a variable *kilos*. The conversion factor is 1 kilogram = 2.2 pounds.

```
>> pounds = 30;
>> kilos = pounds / 2.2
kilos =
   13.6364
```

9) Create a variable *ftemp* to store a temperature in degrees Fahrenheit (F). Convert this to degrees Celsius (C) and store the result in a variable *ctemp*. The conversion factor is C = (F – 32) * 5/9.

```
>> ftemp = 75;
>> ctemp = (ftemp - 32) * 5/9
ctemp =
   23.8889
```

10) Find another quantity to convert from one system of units to another.

```
>> kmPerMile = 1.6093;
>> miles = 6;
>> km = miles * kmPerMile
km =
    9.6558
```

11) The function **sin** calculates and returns the sine of an angle in radians. Use **help elfun** to find the name of the function that returns the sine of an angle in degrees. Verify that calling this function and passing 90 degrees to it results in 1.

```
>> sind(90)
ans =
     1
```

12) A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ. The relationship between them is given by the equations:

```
x = r * cos(θ)
y = r * sin(θ)
```

Assign values for the polar coordinates to variables *r* and *theta*. Then, using these values, assign the corresponding rectangular coordinates to variables *x* and *y*.

```
>> r = 5;
>> theta = 0.5;
>> x = r * cos(theta)
x =
    4.3879
>> y = r * sin(theta)
y =
    2.3971
```

13) Wind often makes the air feel even colder than it is. The Wind Chill Factor (WCF) measures how cold it feels with a given air temperature T (in degrees Fahrenheit) and wind speed (V, in miles per hour). One formula for the WCF is:

$$WCF = 35.7 + 0.6\,T - 35.7\,(V^{0.16}) + 0.43\,T\,(V^{0.16})$$

Create variables for the temperature T and wind speed V, and then using this formula calculate the WCF.

```
>> t = 20;
>> v = 11;
>> wcf = 35.7 + 0.6*t - 35.7*v^0.16+0.43*t*v^0.16
wcf =
    7.9267
```

14) Use **help elfun** or experiment to answer the following questions:

- Is **fix(3.5)** the same as **floor(3.5)**?

```
>> fix(3.5)
ans =
     3
>> floor(3.5)
ans =
     3
```

- Is **fix(3.4)** the same as **fix(-3.4)**?

```
>> fix(3.4)
ans =
     3
>> fix(-3.4)
ans =
```

```
    -3
```

- Is **fix(3.2)** the same as **floor(3.2)**?

```
>> fix(3.2)
ans =
     3
>> floor(3.2)
ans =
     3
```

- Is **fix(-3.2)** the same as **floor(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> floor(-3.2)
ans =
    -4
```

- Is **fix(-3.2)** the same as **ceil(-3.2)**?

```
>> fix(-3.2)
ans =
    -3
>> ceil(-3.2)
ans =
    -3
```

15) Find MATLAB expressions for the following

$\sqrt{19}$

```
sqrt(19)
```

$3^{1.2}$

```
3^1.2
```

```
tan(π)
```

```
tan(pi)
```

16) Use **intmin** and **intmax** to determine the range of values that can be stored in the types **uint32** and **uint64**.

```
>> intmin('uint32')
ans =
           0
>> intmax('uint32')
ans =
  4294967295
>> intmin('uint64')
ans =
                    0
>> intmax('uint64')
ans =
 18446744073709551615
```

17) Are there equivalents to **intmin** and **intmax** for real number types? Use **help** to find out.

```
>> realmin
ans =
  2.2251e-308
>> realmin('double')
ans =
  2.2251e-308
>> realmin('single')
ans =
  1.1755e-38
>> realmax
ans =
  1.7977e+308
```

18) Store a number with a decimal place in a **double** variable (the default). Convert the variable to the type **int32** and store the result in a new variable.

```
>> num = 13.45
num =
   13.4500
>> intnum = int32(num)
intnum =
          13
```

19) Generate a random

- real number in the range from 0 to 1

```
rand
```

- real number in the range from 0 to 20

```
rand * 20
```

- real number in the range from 20 to 50

```
rand*(50-20)+20
```

- integer in the range from 0 to 10

```
round(rand * 10)
```

- integer in the range from 0 to 11

```
round(rand * 11)
```

- integer in the range from 50 to 100

```
round(rand*(100-50)+50)
```

20) Get into a new Command Window, and type **rand** to get a random real number. Make a note of the number. Then, exit MATLAB and repeat this, again making a note of the random number; it should be the same as before. Finally, exit MATLAB and again get into a new Command Window. This time, change the seed before generating a random number; it should be different.

```
>> rand
ans =
0.8147

>> rng('shuffle')
>> rand
ans =
0.4808
```

21) In the ASCII character encoding, the letters of the alphabet are in order: 'a' comes before 'b' and also 'A' comes before 'B'. However, which comes first - lower or uppercase letters?

```
>> int32('a')
ans =
          97
>> int32('A')
ans =
          65
```

```
The upper case letters
```

22) Shift the string ‘xyz’ up in the character encoding by 2 characters.

```
>> char('xyz' + 2)
ans =
z{|
```

23) Using the colon operator, create the following row vectors

```
3     4     5     6

1.0000    1.5000    2.0000    2.5000    3.0000

5     4     3     2

>> 3:6
ans =
     3     4     5     6

>> 1:.5:3
ans =
    1.0000    1.5000    2.0000    2.5000    3.0000

>> 5:-1:2
ans =
     5     4     3     2
```

24) Using the **linspace** function, create the following row vectors

```
4     6     8

-3    -6    -9   -12   -15

9     7     5

>> linspace(4,8,3)
ans =
     4     6     8

>> linspace(-3,-15,5)
ans =
    -3    -6    -9   -12   -15

>> linspace(9,5,3)
ans =
     9     7     5
```

25) Create the following row vectors twice, using **linspace** and using the colon operator:

```
1    2    3    4    5     6     7     8     9    10

>> 1:10
ans =
     1     2     3     4     5     6     7     8     9    10
>> linspace(1,10,10)
ans =
     1     2     3     4     5     6     7     8     9    10


2    7   12

>> 2:5:12
ans =
     2     7    12
>> linspace(2,12,3)
ans =
     2     7    12
```

26) Create a variable *myend* which stores a random integer in the range from 8 to 12. Using the colon operator, create a vector that iterates from 1 to *myend* in steps of 3.

```
>> myend = randi([8,12])
myend =
     8
>> vec = 1:3:myend
vec =
     1     4     7
```

27) Using the colon operator and the transpose operator, create a column vector that has the values -1 to 1 in steps of 0.2.

```
>> rowVec = -1: 0.2: 1;
>> rowVec'
ans =
   -1.0000
   -0.8000
   -0.6000
   -0.4000
   -0.2000
         0
    0.2000
    0.4000
    0.6000
    0.8000
    1.0000
```

28) Write an expression that refers to only the odd-numbered elements in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

```
>> vec = 1:8;
>> vec(1:2:end)
ans =
     1     3     5     7

>> vec = 1:9;
>> vec(1:2:end)
ans =
     1     3     5     7     9
```

29) Create a vector variable *vec*; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether *vec* has an even or odd number of elements (Hint: use a rounding function such as **fix**).

```
>> vec = 1:9;
>> fhalf = vec(1:fix(length(vec)/2))
fhalf =
     1     2     3     4
>> shalf = vec(fix(length(vec)/2)+1:end)
shalf =
     5     6     7     8     9
```

30) Generate a *2 x 3* matrix of random

- real numbers, each in the range from 0 to 1

```
>> rand(2,3)
ans =
    0.0215    0.7369    0.7125
    0.7208    0.4168    0.1865
```

- real numbers, each in the range from 0 to 10

```
>> rand(2,3)*10
ans =
    8.0863    2.2456    8.3067
    2.9409    4.0221    5.0677
```

- integers, each in the range from 5 to 20

```
>> randi([5, 20],2,3)
ans =
```

```
    18    17     5
    11    11     7
```

31) Create a variable *rows* that is a random integer in the range from 1 to 5. Create a variable *cols* that is a random integer in the range from 1 to 5. Create a matrix of all zeros with the dimensions given by the values of *rows* and *cols*.

```
>> rows = randi([1,5])
rows =
     3
>> cols = randi([1,5])
cols =
     2
>> zeros(rows,cols)
ans =
     0     0
     0     0
     0     0
```

32) Find an ***efficient*** way to generate the following matrix:

```
   mat =
         7     8     9    10
        12    10     8     6
```

Then, give expressions that will, for the matrix *mat*,

- refer to the element in the first row, third column
- refer to the entire second row
- refer to the first two columns

```
>> mat = [7:10; 12: -2: 6]
mat =
     7     8     9    10
    12    10     8     6
>> mat(1,3)
ans =
     9
>> mat(2,:)
ans =
    12    10     8     6
>> mat(:,1:2)
ans =
     7     8
    12    10
```

33) Create a 2 x 3 matrix variable *mymat*. Pass this matrix variable to each of the following functions and make sure you understand the result: **fliplr**, **flipud**, and **rot90**. In how many different ways can you **reshape** it?

```
>> mat = randi([1,20], 2,3)
mat =
    16     5     8
    15    18     1
>> fliplr(mat)
ans =
     8     5    16
     1    18    15
>> flipud(mat)
ans =
    15    18     1
    16     5     8
>> rot90(mat)
ans =
     8     1
     5    18
    16    15
>> rot90(rot90(mat))
ans =
     1    18    15
     8     5    16
>> reshape(mat,3,2)
ans =
    16    18
    15     8
     5     1
>> reshape(mat,1,6)
ans =
    16    15     5    18     8     1
>> reshape(mat,6,1)
ans =
    16
    15
     5
    18
     8
     1
```

34) Create a *4 x 2* matrix of all zeros and store it in a variable. Then, replace the second row in the matrix with a vector consisting of a 3 and a 6.

```
>> mymat = zeros(4,2)
mymat =
     0     0
     0     0
     0     0
```

```
     0     0
>> mymat(2,:) = [3 6]
mymat =
     0     0
     3     6
     0     0
     0     0
```

35) Create a vector x which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$. Create a y vector which is **sin(x)**.

```
>> x = linspace(-pi,pi,20);
>> y = sin(x);
```

36) Create a *3 x 5* matrix of random integers, each in the range from -5 to 5. Get the **sign** of every element.

```
>> mat = randi([-5,5], 3,5)
mat =
     5     4     1    -1    -5
     4     4    -1    -3     0
     5    -2     1     0     4
>> sign(mat)
ans =
     1     1     1    -1    -1
     1     1    -1    -1     0
     1    -1     1     0     1
```

37) Create a *4 x 6* matrix of random integers, each in the range from -5 to 5; store it in a variable. Create another matrix that stores for each element the absolute value of the corresponding element in the original matrix.

```
>> mat = randi([-5,5], 4,6)
mat =
    -3    -2    -2     3     4    -1
     1     3    -1     3     0    -2
    -5    -3    -1    -2    -2    -5
    -1     1     5     3    -2     4
>> posmat = abs(mat)
posmat =
     3     2     2     3     4     1
     1     3     1     3     0     2
     5     3     1     2     2     5
     1     1     5     3     2     4
```

38) Create a *3 x 5* matrix of random real numbers. Delete the third row.

```
>> mat = rand(3,5)
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
    0.1730    0.2523    0.1365    0.1991    0.2844

>> mat(3,:) = []
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
```

39) Create a vector variable *vec*. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).

```
>> vec = 1:2:9
vec =
     1     3     5     7     9
>> vec(end)
ans =
     9
>> vec(numel(vec))
ans =
     9
>> vec(length(vec))
ans =
     9
>> v = fliplr(vec)
v =
     9     7     5     3     1
>> v(1)
ans =
     9
```

40) Create a matrix variable *mat*. Find as many expressions as you can that would refer to the last element in the matrix, without assuming that you know how many elements or rows or columns it has (i.e., make your expressions general).

```
>> mat = [12:15; 6:-1:3]
mat =
    12    13    14    15
     6     5     4     3
>> mat(end,end)
ans =
     3
>> mat(end)
ans =
```

```
     3
>> [r c] = size(mat);
>> mat(r,c)
ans =
     3
```

41) Create a three-dimensional matrix and get its **size**.

```
>> mat3d = ones(3,5,2)
mat3d(:,:,1) =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
mat3d(:,:,2) =
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
>> size(mat3d)
ans =
     3     5     2
```

42) The built-in function **clock** returns a vector that contains 6 elements: the first three are the current date (year, month, day) and the last three represent the current time in hours, minutes, and seconds. The seconds is a real number, but all others are integers. Store the result from clock in a variable called *myc*. Then, store the first three elements from this variable in a variable *today* and the last three elements in a variable *now*. Use the fix function on the vector variable *now* to get just the integer part of the current time.

```
>> myc = clock
myc =
   1.0e+03 *
    2.0110    0.0070    0.0100    0.0130    0.0520    0.0537
>> today = myc(1:3)
today =
        2011           7          10
>> now = myc(4:end)
now =
   13.0000   52.0000   53.6860
>> fix(now)
ans =
    13    52    53
```

**Chapter 2: Introduction to MATLAB Programming**

**Exercises**

1) Write a simple script that will calculate the volume of a hollow sphere,

$$\frac{4\pi}{3}(r_o^3 - r_i^3)$$

where $r_i$ is the inner radius and $r_o$ is the outer radius. Assign a value to a variable for the inner radius, and also assign a value to another variable for the outer radius. Then, using these variables, assign the volume to a third variable. Include comments in the script.

Ch2Ex1.m

```
% This script calculates the volume of a hollow sphere

% Assign values for the inner and outer radii
ri = 5.1
ro = 6.8

% Calculate the volume
vol = (4*pi)/3*(ro^3-ri^3)
```

2) The atomic weight is the weight of a mole of atoms of a chemical element. For example, the atomic weight of oxygen is 15.9994 and the atomic weight of hydrogen is 1.0079. Write a script that will calculate the molecular weight of hydrogen peroxide, which consists of two atoms of hydrogen and two atoms of oxygen. Include comments in the script. Use **help** to view the comment in your script.

Ch2Ex2.m

```
% Calculates the molecular weight of hydrogen peroxide

% Initialize the atomic weights for oxygen and hydrogen
atWtOxygen = 15.9994;
atWtHydrogen = 1.0079;

% Hydrogen peroxide is 2 atoms of hydrogen and 2 of oxygen
molWtHydrogenPeroxide = 2*atWtHydrogen + 2 * atWtOxygen
```

```
>> help ch2ex2
 Calculates the molecular weight of hydrogen peroxide
```

3) Write an **input** statement that will prompt the user for the name of a chemical element as a string. Then, find the length of the string.

```
>> elemname = input('Enter a chemical element: ', 's');
Enter a chemical element: hydrogen
>> length(elemname)
ans =
     8
```

4) Write an **input** statement that will prompt the user for a real number, and store it in a variable. Then, use the **fprintf** function to print the value of this variable using 2 decimal places.

```
>> realnum = input('Enter a real number: ');
Enter a real number: 45.789
>> fprintf('The number is %.2f\n', realnum)
The number is 45.79
```

5) The **input** function can be used to enter a vector, such as:

```
>> vec = input('Enter a vector: ')
Enter a vector: 4:7
vec =
            4     5     6     7
```

Experiment with this, and find out how the user can enter a matrix.

```
>> mat = input('Enter a matrix: ')
Enter a matrix: [4:6; 9:11]

mat =
     4     5     6
     9    10    11

>> mat = input('Enter a matrix: ')
Enter a matrix: zeros(2)

mat =
     0     0
     0     0
```

6) Experiment, in the Command Window, with using the **fprintf** function for real numbers. Make a note of what happens for each. Use **fprintf** to print the real number 12345.6789.

```
realnum = 12345.6789;
```

- without specifying any field width

```
>> fprintf('The number is %f\n', realnum)
The number is 12345.678900
```

- in a field width of 10 with 4 decimal places

```
>> fprintf('The number is %10.4f\n', realnum)
The number is 12345.6789
```

- in a field width of 10 with 2 decimal places

```
>> fprintf('The number is %10.2f\n', realnum)
The number is   12345.68
```

- in a field width of 6 with 4 decimal places

```
>> fprintf('The number is %6.4f\n', realnum)
The number is 12345.6789
```

- in a field width of 2 with 4 decimal places

```
>> fprintf('The number is %2.4f\n', realnum)
The number is 12345.6789
```

7) Experiment, in the Command Window, with using the **fprintf** function for integers. Make a note of what happens for each. Use **fprintf** to print the integer 12345.

```
intnum = 12345;
```

- without specifying any field width

```
>> fprintf('The number is %d\n', intnum)
The number is 12345
```

- in a field width of 5

```
>> fprintf('The number is %5d\n', intnum)
The number is 12345
```

- in a field width of 8

```
>> fprintf('The number is %8d\n', intnum)
The number is    12345
```

- in a field width of 3

```
>> fprintf('The number is %3d\n', intnum)
The number is 12345
```

8) Create the following variables

```
x = 12.34;
y = 4.56;
```

Then, fill in the **fprintf** statements using these variables that will accomplish the following:

```
>> fprintf('x is %8.3f\n', x)
```

```
x is   12.340
>> fprintf('x is %.f\n', x)
x is 12
>> fprintf('y is %.1f\n', y)
y is 4.6
>> fprintf('y is %-8.1f!\n', y)
y is 4.6     !
```

9) Write a script to prompt the user for the length and width of a rectangle, and print its area with 2 decimal places. Put comments in the script.

Ch2Ex9.m

```
% Calculate the area of a rectangle

% Prompt the user for the length and width
length = input('Enter the length of the rectangle: ');
width = input('Enter the width of the rectangle: ');

% Calculate and print the area
rect_area = length * width;
fprintf('The area of the rectangle is %.2f\n', rect_area)
```

10) Write a script called *echoname* that will prompt the user for his or her name, and then echo print the name in a sentence in the following format (use %s to print it):

```
>> echoname
What is your name? Susan
Wow, your name is Susan!
```

echoname.m

```
% Prompt the user and echo print name

iname = input('What is your name? ','s');
fprintf('Wow, your name is %s!\n',iname)
```

11) Write a script called *echostring* that will prompt the user for a string, and will echo print the string in quotes:

```
>> echostring
Enter your string: hi there
Your string was: 'hi there'
```

echostring.m

```
% Prompt the user and print a string in quotes

str = input('Enter your string: ', 's');
fprintf('Your string was: ''%s''\n',str)
```

12) In the metric system, fluid flow is measured in cubic meters per second ($m^3/s$). A cubic foot per second ($ft^3/s$) is equivalent to 0.028 $m^3/s$. Write a script titled *flowrate* that will prompt the user for flow in cubic meters per second and will print the equivalent flow rate in cubic feet per second. Here is an example of running the script. Your script must produce output in exactly the same *format* as this:

```
>> flowrate
Enter the flow in m^3/s: 15.2
A flow rate of 15.200 meters per sec
is equivalent to 542.857 feet per sec
```

flowrate.m

```
% Converts a flow rate from cubic meters per second
%  to cubic feet per second

cubMperSec = input('Enter the flow in m^3/s :');
cubFperSec = cubMperSec / .028;
fprintf('A flow rate of %.3f meters per sec\n', ...
   cubMperSec)
fprintf('is equivalent to %.3f feet per sec\n', ...
   cubFperSec)
```

13) On average, people in a region spend 8% to 10% of their income on food. Write a script that will prompt the user for an annual income. It will then print the range that would typically be spent on food annually. Also, print a monthly range.

Ch2Ex13.m

```
% Calculates and prints the likely $ amount spent on food
% based on annual income

income = input('Enter your annual income: ');
fprintf('You are likely to spend between $%.2f\n',.08*income)
fprintf('and $%.2f annually on food.\n\n', .1*income)

% print the monthly range
fprintf('You are likely to spend between $%.2f\n',.08*income/12)
fprintf('and $%.2f monthly on food.\n', .1*income/12)
```

14) Wing loading, which is the airplane weight divided by the wing area, is an important design factor in aeronautical engineering. Write a script that will prompt the user for the weight of the aircraft in kilograms, and the wing area in meters squared, and will calculate and print the wing loading of the aircraft in kilograms per square meter.

Ch2Ex14.m

```
% Calculates the wing loading for an airplane
```

```
% Prompt the user for the weight and wing area of the plane
plane_weight = input('Enter the weight of the airplane: ');
wing_area = input('Enter the wing area: ');

% Calculate and print the wing loading
fprintf('The wing loading is %.2f\n', plane_weight/wing_area)
```

15) Write a script that assigns values for the x coordinate and then y coordinate of a point, and then plots this using a green +.

Ch2Ex15.m

```
% Prompt the user for the coordinates of a point and plot
%  the point using a green +

x = input('Enter the x coordinate: ');
y = input('Enter the y coordinate: ');

plot(x,y, 'g+')
```

16) Plot **exp(x)** for values of x ranging from -2 to 2 in steps of 0.1. Put an appropriate title on the plot, and label the axes.

Ch2Ex16.m

```
% Plots exp(x)

x = -2:0.1:2;
y = exp(x);
plot(x,y,'*')
title('Exp(x)')
xlabel('x')
ylabel('exp(x)')
```

17) Create a vector *x* with values ranging from 1 to 100 in steps of 5. Create a vector *y* which is the square root of each value in x. Plot these points. Now, use the **bar** function instead of **plot** to get a bar chart instead.

Ch2Ex17.m

```
% Plots same x and y points using bar and plot

clf
x = 1:5:100;
y = sqrt(x);
plot(x,y,'*')
title('sqrt(x)')
```

```
figure(2)
bar(x,y)
title('sqrt(x)')
```

18) Create a *y* vector which stores random integers in the 1 to 100 range. Create an *x* vector which iterates from 1 to the length of the *y* vector. Experiment with the **plot** function using different colors, line types, and plot symbols.

Ch2Ex18.m

```
% Experiment with plot symbols and line types

y = randi([1, 100],1,15)
x = 1:length(y);
clf
plot(x,y,'rv')
figure(2)
plot(x,y,'g*:')
```

19) Plot **sin(x)** for x values ranging from 0 to π (in separate Figure Windows):

- using 10 points in this range
- using 100 points in this range

Ch2Ex19.m

```
% Plots sin(x) with 10 points and 100 points in range 0 to pi

x = linspace(0,pi,10);
y = sin(x);
clf
figure(1)
plot(x,y,'k*')
title('sin(x) with 10 points')
figure(2)
x = linspace(0,pi);
y = sin(x);
plot(x,y,'k*')
title('sin(x) with 100 points')
```

20) Atmospheric properties such as temperature, air density, and air pressure are important in aviation. Create a file that stores temperatures in degrees Kelvin at various altitudes. The altitudes are in the first column and the temperatures in the second. For example, it may look like this:

```
 1000         288
 2000         281
 3000         269
 5000         256
10000         223
```

Write a script that will load this data into a matrix, separate it into vectors, and then plot the data with appropriate axis labels and a title.

Ch2Ex20.m

```
% Read altitudes and temperatures from a file and plot

load alttemps.dat
altitudes = alttemps(:,1);
temps = alttemps(:,2);
plot(altitudes,temps,'k*')
xlabel('Altitudes')
ylabel('Temperatures')
title('Atmospheric Data')
```

21) Create a *3 x 6* matrix of random integers, each in the range from 50 to 100. Write this to a file called *randfile.dat*. Then, create a new matrix of random integers, but this time make it a *2 x 6* matrix of random integers, each in the range from 50 to 100. Append this matrix to the original file. Then, read the file in (which will be to a variable called *randfile*) just to make sure that worked!

```
>> mat = randi([50,100], 3,6)
mat =
    91    96    64    99    98    57
    96    82    77    58    74    71
    56    54    98    99    90    96
>> save randfile.dat mat -ascii
>> newmat = randi([50,100], 2,6)
newmat =
    90    83    93    84    87    83
    98    51    97    88    70    58
>> save randfile.dat newmat -ascii -append
>> load randfile.dat
>> randfile
randfile =
    91    96    64    99    98    57
    96    82    77    58    74    71
    56    54    98    99    90    96
    90    83    93    84    87    83
    98    51    97    88    70    58
```

22) Create a file called "testtan.dat" comprised of two lines with three real numbers on each line (some negative, some positive, in the -1 to 3 range). The file can be created from the Editor, or saved from a matrix. Then, **load** the file into a matrix and calculate the tangent of every element in the resulting matrix.

```
>> mat = rand(2,3)*4-1
```

```
mat =
    1.8242    0.1077   -0.6115
   -0.8727   -0.8153    2.2938
>> save testtan.dat mat -ascii
>> load testtan.dat
>> tan(testtan)
ans =
   -3.8617    0.1081   -0.7011
   -1.1918   -1.0617   -1.1332
```

23) Write a function *calcrectarea* that will calculate and return the area of a rectangle. Pass the length and width to the function as input arguments.

calcrectarea.m

```
function area = calcrectarea(length, width)
% This function calculates the area of a rectangle
% Format of call: calcrectarea(length, width)
% Returns the area

area = length * width;
end
```

24) Write a function called *fn* that will calculate y as a function of x, as follows:

$$y = x^3 - 4x^2 + \sin(x)$$

Here are two examples of using the function:

```
>> help fn
  Calculates y as a function of x

>> y = fn(7)
y =
  147.6570
```

fn.m

```
function out = fn(x)
% Calculates y as a function of x

% Format of call: fn(x)
% Returns x^2-4*x^2+sin(x)

out = x^3 - 4*x^2 + sin(x);
end
```

*Renewable energy sources such as biomass are gaining increasing attention. Biomass energy units include megawatt hours (MWh) and gigajoules (GJ). One MWh is equivalent to 3.6 GJ. For example, one cubic meter of wood chips produces 1 MWh.*

25) Write a function *mwh_to_gj* that will convert from MWh to GJ. Here are some examples of using the function.

```
>> mwh = 3.3;
>> gj = mwh_to_gj(mwh)
gj =
   11.8800

>> disp(mwh_to_gj(1.1))
    3.9600

>> help mwh_to_gj
  Converts from MWh to GJ
```

mwh_to_gj.m

```
function out = mwh_to_gj(mwh)
% Converts from MWh to GJ

% Format of call: mwh_to_gj(mwh)
% Returns gigajoules

out = mwh * 3.6;
end
```

26) The velocity of an aircraft is typically given in either miles/hour or meters/second. Write a function that will receive one input argument, the velocity of an airplane in miles per hour and will return the velocity in meters per second. The relevant conversion factors are: one hour = 3600 seconds, one mile = 5280 feet, and one foot = .3048 meters.

convertVel.m

```
function velmps = convertVel(velmph)
% Convert the velocity of an aircraft from
% miles per hour to meters per second
% Format of call: convertVel(mph)
% Returns velocity in mps

velmps = velmph/3600*5280*.3048;
end
```

27) If a certain amount of money (called the principal P) is invested in a bank account, earning an interest rate i compounded annually, the total amount of money $T_n$ that will be in the account after n years is given by:

$$T_n = P\ (1 + i)^n$$

Write a function that will receive input arguments for P, i, and n, and will return the total amount of money $T_n$. Also, give an example of calling the function.

```
function totMoney = account(p,i,n)
% Calculates the amount of money in an account
% after n years at interest rate i with a
% principal p invested
% Format of call: account(p,i,n)
% Returns total of money after n years

totMoney = p * (1+i)^n;
end
```

```
>> format bank
>> total = account(50000,0.05,10)
total =
      81444.73
```

28) List some differences between a script and a function.

- A function has a header whereas a script does not.
- A function typically has end at the **end** of the file.
- A function is called whereas a script is executed.
- Arguments are passed to functions but not to scripts.
- Functions can return arguments whereas scripts cannot.
- The block comment is typically in the beginning of a script but under the function header.

29) The velocity of a moving fluid can be found from the difference between the total and static pressures $P_t$ and $P_s$. For water, this is given by

$$V = 1.016\ \sqrt{P_t - P_s}$$

Write a function that will receive as input arguments the total and static pressures and will return the velocity of the water.

fluidvel.m

```
function waterVel = fluidvel(totp, statp)
% Calculates the velocity of water given the
% total and static pressures
% Format: fluidvel(total pressure, static pressure)
% Returns the velocity of the water

waterVel = 1.016 * sqrt(totp-statp);
```

```
end
```

30) For a project, some biomedical engineering students are designing a device that will monitor a person's heart rate while on a treadmill. The device will let the subject know when the target heart rate has been reached. A simple calculation of the target heart rate (THR) for a moderately active person is

$$\text{THR} = (220 - A) * .6$$

where A is the person's age. Write a function that will calculate and return the THR.

findthr.m

```
function thr = findthr(age)
% Calculate the target heart rate for
% a person of a given age
% Format of call: findthr(age)
% Returns target heart rate

thr = (220 - age) * 0.6;
end
```

31) An approximation for a factorial can be found using Stirling's formula:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Write a function to implement this, passing the value of n as an argument.

stirling.m

```
function approxNFact = stirling(n)
% Approximate n! using Stirling's formula
% Format of call: stirling(n)
% Returns approximate factorial of n

approxNFact = sqrt(2*pi*n) * (n/exp(1))^n;
end
```

32) The cost of manufacturing n units (where n is an integer) of a particular product at a factory is given by the equation:

$$C(n) = 5n^2 - 44n + 11$$

Write a script *mfgcost* that will

- prompt the user for the number of units n
- call a function *costn* that will calculate and return the cost of manufacturing n units
- print the result (the format must be exactly as shown below)

Next, write the function *costn,* which simply receives the value of n as an input argument, and calculates and returns the cost of manufacturing n units.

Here is an example of executing the script:

```
>> mfgcost
Enter the number of units: 100
The cost for 100 units will be $45611.00
>>
```

mfgcost.m

```
% Prompts the user for the number of units, calls a function
% to calculate the cost for producing that many units, and
% prints this result

n = input('Enter the number of units: ');
costPerN = costn(n);
fprintf('The cost for %d units will be ',n)
fprintf('$%.2f\n', costPerN)
```

costn.m

```
function out = costn(n)
% Calculates the cost of manufacturing n
%  units of a product
% Format of call: costn(n)
% Returns cost as 5*n^2-44*n+11

out = 5*n^2 - 44*n + 11;
end
```

33) The conversion depends on the temperature and other factors, but an approximation is that 1 inch of rain is equivalent to 6.5 inches of snow. Write a script that prompts the user for the number of inches of rain, calls a function to return the equivalent amount of snow, and prints this result. Write the function, as well!

Ch2Ex33.m

```
% Prompt the user for a number of inches of rain
% and call a function to calculate the
% equivalent amount of snow

rain = input('How much rain in inches: ');
snow = rainToSnow(rain);
fprintf('%.1f inches of rain would be ', rain)
fprintf('%.1f inches of snow\n', snow)
```

rainToSnow.m

```
function outsnow = rainToSnow(rain)
% Calculate equivalent amount of snow
% given rainfall in inches
% Format of call: rainToSnow(rain)
```

```
% Returns equivalent snowfall

outsnow = rain * 6.5;
end
```

34) The volume V of a regular tetrahedron is given by $V = \frac{1}{12}\sqrt{2}\ s^3$ where s is the length of the sides of the equilateral triangles that form the faces of the tetrahedron. Write a program to calculate such a volume. The program will consist of one script and one function. The function will receive one input argument, which is the length of the sides, and will return the volume of the tetrahedron. The script will prompt the user for the length of the sides, call the function to calculate the volume, and print the result in a nice sentence format. For simplicity, we will ignore units.

Ch2Ex34.m

```
% Calculates and prints the volume of a regular tetrahedron
% Prompts user for length of sides, calls a function to
% calculate the volume, and prints the result

len = input('Enter the length of the sides: ');
volume = tetVol(len);
fprintf('The volume of a regular tetrahedron with ')
fprintf(' sides of length %.1f is %.2f\n', len, volume)
```

tetVol.m

```
function vol = tetVol(len)
% Calculates the volume of a regular tetrahedron
% Format of call: tetVol(side length)
% Returns volume

vol = 1/12 * sqrt(2) * len^3;
end
```

35) Write a function that is called *pickone,* which will receive one input argument x ,which is a vector, and will return one random element from the vector. For example,

```
>> pickone(4:7)
ans =
     5

>> disp(pickone(-2:0))
-1

>> help pickone
pickone(x) returns a random element from vector x
```

pickone.m

```
function elem = pickone(invec)
% pickone(x) returns a random element from vector x
% Format of call: pickone(vector)
% Returns random element from the vector

len = length(invec);
ran = randi([1, len]);
elem = invec(ran);
end
```

36) A function can return a vector as a result. Write a function *vecout* that will receive one integer argument and will return a vector that increments from the value of the input argument to its value plus 5, using the colon operator. For example,

```
>> vecout(4)
ans =

     4     5     6     7     8     9
```

vecout.m

```
function outvec = vecout(innum)
% Create a vector from innum to innum + 5
% Format of call: vecout(input number)
% Returns a vector input num : input num+5

outvec = innum:innum+5;
end
```

37) If the lengths of two sides of a triangle and the angle between them are known, the length of the third side can be calculated. Given the lengths of two sides (b and c) of a triangle, and the angle between them α in degrees, the third side a is calculated as follows:

$a^2 = b^2 + c^2 - 2\ b\ c\ \cos(\alpha)$

Write a script *thirdside* that will prompt the user and read in values for b, c, and α (in degrees), and then calculate and print the value of a with 3 decimal places. (Note: To convert an angle from degrees to radians, multiply the angle by $\pi/180$). The format of the output from the script should look exactly like this:

```
>> thirdside
Enter the first side: 2.2
Enter the second side: 4.4
Enter the angle between them: 50

The third side is 3.429
```

For more practice, write a function to calculate the third side, so the script will call this function.

thirdside.m

```
% Calculates the third side of a triangle, given
% the lengths of two sides and the angle between them

b = input('Enter the first side: ');
c = input('Enter the second side: ');
alpha = input('Enter the angle between them: ');
alpha = alpha * pi / 180;

a = sqrt(b^2 + c^2 -2*b*c*cos(alpha));
fprintf('\nThe third side is %.3f\n', a)
```

38) A part is being turned on a lathe. The diameter of the part is supposed to be 20,000 mm. The diameter is measured every 10 minutes and the results are stored in a file called *partdiam.dat*. Create a data file to simulate this. The file will store the time in minutes and the diameter at each time. Plot the data.

partdiam.dat

```
0  25233
10 23432
20 21085
30 20374
40 20002
```

Ch2Ex38.m

```
% Read from a data file the diameter of a part every 10 minutes
% as it is turned on a lathe and plot this data

load partdiam.dat
mins = partdiam(:,1);
diams = partdiam(:,2);
plot(mins,diams,'k*')
xlabel('minutes')
ylabel('part diameter')
```

39) A file "floatnums.dat" has been created for use in an experiment. However, it contains float (real) numbers and what is desired instead is integers. Also, the file is not exactly in the correct format; the values are stored columnwise rather than rowwise. For example, if the file contains the following:

```
   90.5792   27.8498   97.0593
   12.6987   54.6882   95.7167
   91.3376   95.7507   48.5376
```

```
   63.2359   96.4889   80.0280
    9.7540   15.7613   14.1886
```

what is really desired is:

```
    91    13    91    63    10
    28    55    96    96    16
    97    96    49    80    14
```

Create the data file in the specified format. Write a script that would read from the file *floatnums.dat* into a matrix, round the numbers, and write the matrix in the desired format to a new file called *intnums.dat*.

Ch2Ex39.m

```
% Reads in float numbers stored column-wise from a file,
% rounds them to integers and writes row-wise to a new file

load floatnums.dat
inums = round(floatnums)';
save intnums.dat inums -ascii
```

40) A file called *costssales.dat* stores for a company some cost and sales figures for the last n quarters (n is not defined ahead of time). The costs are in the first column, and the sales are in the second column. For example, *if* five quarters were represented, there would be five lines in the file, and it might look like this:

```
1100      800
1233      650
1111      1001
1222      1300
 999      1221
```

Write a script called *salescosts* that will read the data from this file into a matrix. When the script is executed, it will do three things. First, it will print how many quarters were represented in the file, such as:

```
>> salescosts
There were 5 quarters in the file
```

Next, it will plot the costs using black circles and sales using black stars (*) in a Figure Window with a legend (using default axes) as seen in Figure 2.8.

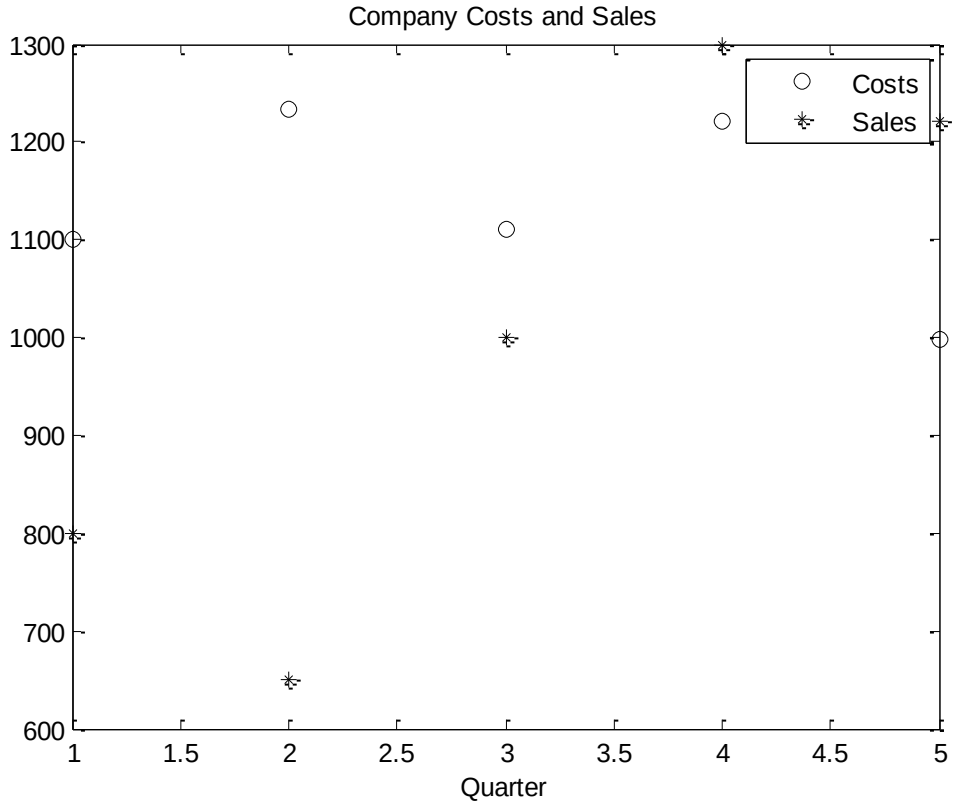



Figure 2.8 Plot of Cost and Sales data

Finally, the script will write the data to a new file called *newfile.dat* in a different order. The sales will be the first row, and the costs will be the second row. For example, if the file is as shown above, the resulting file will store the following:

```
800         650         1001         1300         1221
1100        1233         1111         1222          999
```

It should not be assumed that the number of lines in the file is known.

salescosts.m

```
load costssales.dat

costs = costssales(:,1);
sales = costssales(:,2);

len = length(costs); % or sales

x = 1:len;  % Note: not necessary
plot(x,costs,'ko')
hold on
plot(x,sales,'k*')
legend('Costs', 'Sales')
fprintf('There were %d quarters in the file\n', len)
neworder = rot90(costssales)
```

```
% neworder = flipud(costssales');
save newfile.dat neworder -ascii
```