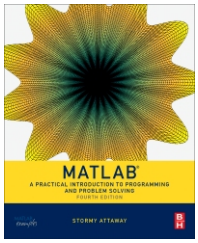
MATLAB: A Practical Introduction to Programming and Problem Solving

Fourth Edition

SOLUTION MANUAL



Stormy Attaway

College of Engineering
Boston University

Chapter 1: Introduction to MATLAB

Exercises

1) Create a variable *myage* and store your age in it. Subtract 2 from the value of the variable. Add 1 to the value of the variable. Observe the Workspace Window and Command History Window as you do this.

```
>> myage = 20;
>> myage = myage - 2;
>> myage = myage + 1;
```

2) Explain the difference between these two statements:

```
result = 9*2
result = 9*2;
```

Both will store 18 in the variable result. In the first, MATLAB will display this in the Command Window; in the second, it will not.

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) Create two variables to store a weight in pounds and ounces. Use **who** and **whos** to see the variables. Use **class** to see the types of the variables. Clear one of them and then use **who** and **whos** again.

```
>> pounds = 4;
>> ounces = 3.3;
>> who

Your variables are:

ounces  pounds

>> whos
  Name         Size              Bytes  Class
Attributes

  ounces       1x1                   8  double
  pounds       1x1                   8  double
```

```
>> clear pounds
>> who

Your variables are:

ounces
```

5) 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
```

6) 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
```

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

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

>> 25/5*5
ans =
```

```
    25
>> 4 + 3 ^ 2
ans =
    13
>> (4 + 3) ^ 2
ans =
    49
>> 3 \ 12 + 5
ans =
     9
>> 4 - 2 * 3
ans =
    -2
```

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 lb.

```
>> 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) The following assignment statements either contain at least one error, or could be improved in some way. Assume that *radius* is a variable that has been initialized. First, identify the problem, and then fix and/or improve them:

```
33 = number

     The variable is always on the left
     number = 33
```

```
my variable =  11.11;

     Spaces are not allowed in variable names
     my_variable = 11.11;

area = 3.14 * radius^2;

     Using pi is more accurate than 3.14
     area = pi * radius^2;

x = 2 * 3.14 * radius;

     x is not a descriptive variable name
     circumference  =  2 * pi * radius;
```

11) Experiment with the functional form of some operators such as **plus**, **minus**, and **times**.

```
>> plus(4, 8)
ans =
    12
>> plus(3, -2)
ans =
     1
>> minus(5, 7)
ans =
    -2
>> minus(7, 5)
ans =
     2
>> times(2, 8)
ans =
    16
```

12) Generate a random

- real number in the range (0, 20)

```
rand * 20
```

- real number in the range (20, 50)

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

- integer in the inclusive range from 1 to 10

```
randi(10)
```

- integer in the inclusive range from 0 to 10

```
randi([0, 10])
```

- integer in the inclusive range from 50 to 100

```
randi([50, 100])
```

13) 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
```

14) What is the difference between x and ‘x’?

```
In an expression, the first would be interpreted as the name
of a variable, whereas ‘x’ is the character x.
```

15) What is the difference between 5 and ‘5’?

```
The first is the number 5, the second is the character 5.
(Note: int32(5) is 53.  So, 5+1 would be 6.  ‘5’+1 would be
54.)
```

16) 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
```

17) Explain the difference between constants and variables.

Constants store values that are known and do not change. Variables are used when the value will change, or when the value is not known to begin with (e.g., the user will provide the value).

18) What would be the result of the following expressions?

```
'b' >= 'c' - 1          1

3 == 2 + 1              1

(3 == 2) + 1            1

xor(5 < 6, 8 > 4)       0

10 > 5 > 2

  0  Evaluated from left to right: 10>5 is 1,
     then 1 > 2 is 0

result = 3^2 - 20;
0 <= result <= 10

     1    Evaluated left to right: 0 <= result is 0,
     then 0 <= 10 is 1
```

19) Create two variables *x* and *y* and store numbers in them. Write an expression that would be **true** if the value of *x* is greater than five or if the value of *y* is less than ten, but not if both of those are **true**.

```
>> x = 3;
>> y = 12;
>> xor(x > 5, y < 10)
ans =
     0
```

20) Use the equality operator to verify that 3*10^5 is equal to 3e5.

```
>> 3*10^5 == 3e5
```

```
ans =
     1
```

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) 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
```

23) 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
```

24) Use the **cast** function to cast a variable to be the same type as another variable.

```
>> vara = uint16(3 + 5)
vara =
      8
>> varb = 4*5;
>> class(varb)
ans =
double
>> varb = cast(varb, 'like', vara)
varb =
     20
>> class(varb)
ans =
uint16
```

25) 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
```

26) For what range of values is the function **round** equivalent to the function **floor**?
For positive numbers: when the decimal part is less than .5
For negative numbers: when the decimal part is greater than or equal to .5

For what range of values is the function **round** equivalent to the function **ceil**?
For positive numbers: when the decimal part is greater than or equal to .5
For negative numbers: when the decimal part is less than .5

27) Use **help** to determine the difference between the **rem** and **mod** functions.

```
>> help rem
 rem    Remainder after division.
    rem(x,y) is x - n.*y where n = fix(x./y) if y ~= 0.
    By convention:
       rem(x,0) is NaN.
       rem(x,x), for x~=0, is 0.
       rem(x,y), for x~=y and y~=0, has the same sign as x.

rem(x,y) and MOD(x,y) are equal if x and y have the same
sign, but differ by y if x and y have different signs.

>> help mod
 mod    Modulus after division.
```

```
mod(x,y) is x - n.*y where n = floor(x./y) if y ~= 0.

By convention:
   mod(x,0) is x.
   mod(x,x) is 0.
   mod(x,y), for x~=y and y~=0, has the same sign as y.
```

28) Find MATLAB expressions for the following

$\sqrt{19}$

```
sqrt(19)
```

$3^{1.2}$

```
3^1.2
```

$\tan(\pi)$

```
tan(pi)
```

29) Using only the integers 2 and 3, write as many expressions as you can that result in 9. Try to come up with at least 10 different expressions (e.g., don't just change the order). Be creative! Make sure that you write them as MATLAB expressions. Use operators and/or built-in functions.

```
3 ^ 2

2 ^ 3 + (3 - 2)

3 * 3

3 ^ 3 - 3 * 3 * 2

2^3 + abs(2-3)

2^3 + sign(3)

3/2*2*3

2\3*2*3

sqrt(3^(2+2))
```

```
nthroot(3^(2+2),2)
```

30) A vector can be represented by its rectangular coordinates x and y or by its polar coordinates r and θ. Theta is measured in radians. 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
```

31) In special relativity, the Lorentz factor is a number that describes the effect of speed on various physical properties when the speed is significant relative to the speed of light. Mathematically, the Lorentz factor is given as:

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

Use 3×10^8 m/s for the speed of light, *c*. Create variables for *c* and the speed *v* and from them a variable *lorentz* for the Lorentz factor.

```
>> c = 3e8;
>> v = 2.9e8;
>> lorentz = 1 / sqrt(1 - v^2/c^2)
lorentz =
    3.9057
```

32) A company manufactures a part for which there is a desired weight. There is a tolerance of N percent, meaning that the range between minus and plus N% of the desired weight is acceptable. Create a variable that stores a weight, and another variable for N (for example, set it to two). Create variables that store the minimum and maximum values in the acceptable range of weights for this part.

```
>> weight = 12.3;
>> N = 2;
```

```
>> min = weight - weight*0.01*N
min =
   12.0540
>> max = weight + weight*0.01*N
max =
   12.5460
```

33) An environmental engineer has determined that the cost C of a containment tank will be based on the radius *r* of the tank:

$$C = \frac{32430}{r} + 428\pi r$$

Create a variable for the radius, and then for the cost.

```
>> format bank
>> radius = 11;
>> cost = 32430/radius + 428*pi*radius
cost =
      17738.80
```

34) A chemical plant releases an amount A of pollutant into a stream. The maximum concentration C of the pollutant at a point which is a distance x from the plant is:

$$C = \frac{A}{x}\sqrt{\frac{2}{\Pi e}}$$

Create variables for the values of A and x, and then for C. Assume that the distance x is in meters. Experiment with different values for x.

```
>> A = 30000;
>> x = 100;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
        145.18
>> x = 1000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
         14.52
>> x = 20000;
>> C = A/x * sqrt(2/(pi*exp(1)))
C =
          0.73
```

35) The geometric mean g of n numbers x_i is defined as the n^{th} root of the product of x_i:

$$g = \sqrt[n]{x_1 x_2 x_3 \ldots x_n}$$

(This is useful, for example, in finding the average rate of return for an investment which is something you'd do in engineering economics). If an investment returns 15% the first year, 50% the second, and 30% the third year, the average rate of return would be `(1.15*1.50*1.30)^(1/3)`.) Compute this.

```
>> x1 = 1.15;
>> x2 = 1.5;
>> x3 = 1.3;
>> gmean = nthroot(x1*x2*x3, 3)
gmean =
          1.31
```

36) Use the **deg2rad** function to convert 180 degrees to radians.

```
>> deg2rad(180)
ans =
    3.1416
>>
```

Chapter 2: Vectors and Matrices

Exercises

1) If a variable has the dimensions 3 x 4, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

2) If a variable has the dimensions 1 x 5, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

3) If a variable has the dimensions 5 x 1, could it be considered to be (**bold** all that apply):
a matrix
a row vector

a column vector
a scalar

4) If a variable has the dimensions 1 x 1, could it be considered to be (**bold** all that apply):
a matrix
a row vector
a column vector
a scalar

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

```
     2     3     4     5     6     7

     1.1000    1.3000    1.5000    1.7000

     8     6     4     2
```

```
>> 2:7
ans =
     2     3     4     5     6     7
>> 1.1:0.2:1.7
ans =
    1.1000    1.3000    1.5000    1.7000
>> 8:-2:2
ans =
     8     6     4     2
```

6) Using a built-in function, create a vector *vec* which consists of 20 equally spaced points in the range from –pi to +pi.

```
vec = linspace(0,2*pi,50);
```

7) Write an expression using **linspace** that will result in the same as 2: 0.2: 3

```
linspace(2,3,6)
```

8) Using the colon operator and also the **linspace** function, create the following row vectors:

```
    -5    -4    -3    -2    -1

     5     7     9

     8     6     4
```

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

9) How many elements would be in the vectors created by the following expressions?

```
linspace(3,2000)

      100 (always, by default)

logspace(3,2000)

      50 (always, by default – although these numbers
            would get very large quickly; most would be
      represented as Inf)
```

10) Create a variable *myend* which stores a random integer in the inclusive range from 5 to 9. Using the colon operator, create a vector that iterates from 1 to *myend* in steps of 3.

```
>> myend = randi([5, 9])
myend =
     8
>> vec = 1:3:myend
vec =
     1     4     7
```

11) Using the colon operator and the transpose operator, create a column vector *myvec* that has the values -1 to 1 in steps of 0.5.

```
>> rowVec = -1: 0.5: 1;
```

```
>> rowVec'
ans =
   -1.0000
   -0.5000
         0
    0.5000
    1.0000
```

12) Write an expression that refers to only the elements that have odd-numbered subscripts 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 = 4:12
vec =
     4     5     6     7     8     9    10    11    12
>> vec(1:2:end)
ans =
     4     6     8    10    12
```

13) Generate a 2 x 4 matrix variable *mat*. Replace the first row with 1:4. Replace the third column (you decide with which values).

```
>> mat = [2:5; 1 4 11 3]
mat =
     2     3     4     5
     1     4    11     3
>> mat(1,:) = 1:4
mat =
     1     2     3     4
     1     4    11     3
>> mat(:,3) = [4;3]
mat =
     1     2     4     4
     1     4     3     3
```

14) Generate a 2 x 4 matrix variable *mat*. Verify that the number of elements is the product of the number of rows and columns.

```
>> mat = randi(20,2,4)
mat =
     1    19    17     9
```

```
    13    15    20    16
>> [r c] = size(mat);
>> numel(mat) == r * c
ans =
     1
```

15) Which would you normally use for a matrix: **length** or **size**? Why?
Definitely **size**, because it tells you both the number of rows and columns.

16) When would you use **length** vs. **size** for a vector?
If you want to know the number of elements, you'd use **length**. If you want to figure out whether it's a row or column vector, you'd use **size**.

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

- real numbers, each in the range (0, 1)

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

- real numbers, each in the range (0, 10)

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

- integers, each in the inclusive range from 5 to 20

```
>> randi([5, 20],2,3)
ans =
    18    17     5
    11    11     7
```

18) Create a variable *rows* that is a random integer in the inclusive range from 1 to 5. Create a variable *cols* that is a random integer in the inclusive 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
```

19) 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
```

20) 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(1)
ans =
```

```
     9
```

21) Create a 2 x 3 matrix variable *mat*. Pass this matrix variable to each of the following functions and make sure you understand the result: **flip**, **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
>> flip(mat)
ans =
    15    18     1
    16     5     8
>>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
```

22) What is the difference between `fliplr(mat)` and `mat = fliplr(mat)`?

The first stores the result in *ans* so *mat* is not changed; the second changes *mat*.

23) Use **reshape** to reshape the row vector 1:4 into a 2x2 matrix; store this in a variable named mat. Next, make 2x3 copies of mat using both **repelem** and **repmat**.

```
>> mat = reshape(1:4,2,2)
mat =
     1     3
     2     4
>> repelem(mat,2,3)
ans =
     1     1     1     3     3     3
     1     1     1     3     3     3
     2     2     2     4     4     4
     2     2     2     4     4     4
>> repmat(mat,2,3)
ans =
     1     3     1     3     1     3
     2     4     2     4     2     4
     1     3     1     3     1     3
     2     4     2     4     2     4
```

24) 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
```

25) Given the matrix:

```
>> mat = randi([1 20], 3,5)
mat =
     6    17     7    13    17
```

```
    17     5     4    10    12
     6    19     6     8    11
```

Why wouldn't this work:

```
mat(2:3, 1:3) = ones(2)
```

Because the left and right sides are not the same dimensions.

26) Create a three-dimensional matrix with dimensions 2 x 4 x 3 in which the first "layer" is all 0s, the second is all 1s and the third is all 5s. Use **size** to verify the dimensions.

```
>> mat3d = zeros(2,4,3);
>> mat3d(:,:,2) = 1;
>> mat3d(:,:,3) = 5;
>> mat3d
mat3d(:,:,1) =
     0     0     0     0
     0     0     0     0
mat3d(:,:,2) =
     1     1     1     1
     1     1     1     1
mat3d(:,:,3) =
     5     5     5     5
     5     5     5     5
```

27) 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);
```

28) Create a *3 x 5* matrix of random integers, each in the inclusive 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
```

29) Find the sum 3+5+7+9+11.

```
>> sum(3:2:11)
ans =
    35
```

30) Find the sum of the first n terms of the harmonic series where n is an integer variable greater than one.

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\ldots$$

```
>> n = 4;
>> sum(1./(1:n))
ans =
    2.0833
```

31) Find the following sum by first creating vectors for the numerators and denominators:

$$\frac{3}{1}+\frac{5}{2}+\frac{7}{3}+\frac{9}{4}$$

```
>> num = 3:2:9
num =
     3     5     7     9
>> denom = 1:4
denom =
     1     2     3     4
>> fracs = num ./ denom
fracs =
    3.0000    2.5000    2.3333    2.2500
>> sum(fracs)
ans =
   10.0833
```

32) Create a matrix and find the product of each row and column using **prod**.

```
>> mat = randi([1, 30], 2,3)
mat =
    11    24    16
     5    10     5
```

```
>> prod(mat)
ans =
    55   240    80

>> prod(mat,2)
ans =
        4224
         250
```

33) Create a *1 x 6* vector of random integers, each in the inclusive range from 1 to 20. Use built-in functions to find the minimum and maximum values in the vector. Also create a vector of cumulative sums using **cumsum**.

```
vec = randi([1,20], 1,6)
min(vec)
max(vec)
cvec = cumsum(vec)
```

34) Write a relational expression for a vector variable that will verify that the last value in a vector created by **cumsum** is the same as the result returned by **sum**.

```
>> vec = 2:3:17
vec =
     2     5     8    11    14    17
>> cv = cumsum(vec)
cv =
     2     7    15    26    40    57
>> sum(vec) == cv(end)
ans =
     1
```

35) Create a vector of five random integers, each in the inclusive range from -10 to 10. Perform each of the following:

```
>> vec = randi([-10, 10], 1,5)
```

- subtract 3 from each element

```
>> vec-3
```

- count how many are positive

```
>> sum(vec > 0)
```

- get the cumulative minimum

36) Create a *3 x 5* matrix. Perform each of the following:

```
>> mat = randi([-10 10], 3,5)
```

- Find the maximum value in each column.

```
>> max(mat)
```

- Find the maximum value in each row.

```
>> max(mat, [], 2)
>> max(mat')
```

- Find the maximum value in the entire matrix.

```
>> max(max(mat))
```

- Find the cumulative maxima.

```
>> cummax(mat)
```

37) Find two ways to create a *3 x 5* matrix of all 100s (Hint: use **ones** and **zeros**).

```
ones(3,5)*100
```

```
zeros(3,5)+100
```

38) Given the two matrices:

A

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & -1 & 6 \end{bmatrix}$$

B

$$\begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & 0 \end{bmatrix}$$

Perform the following operations:

A + B

$$\begin{bmatrix} 3 & 6 & 4 \\ 5 & 2 & 6 \end{bmatrix}$$

A – B

$$\begin{bmatrix} -1 & -2 & 2 \\ 3 & -4 & 6 \end{bmatrix}$$

A .* B

$$\begin{bmatrix} 2 & 8 & 3 \\ 4 & -3 & 0 \end{bmatrix}$$

39) 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 *
    2.0130    0.0010    0.0080    0.0120    0.0060
0.0014
>> today = myc(1:3)
today =
        2013           1           8
>> now = myc(4:end)
now =
   12.0000    6.0000    1.4268
>> fix(now)
ans =
    12     6     1
```

40) A vector *v* stores for several employees of the Green Fuel Cells Corporation their hours worked one week followed for each by the hourly pay rate. For example, if the variable stores

```
>> v
v =
33.0000  10.5000  40.0000  18.0000  20.0000  7.5000
```

that means the first employee worked 33 hours at $10.50 per hour, the second worked 40 hours at $18 an hour, and so on. Write code that will separate this into two vectors, one that stores the hours worked and another that stores the hourly rates. Then, use the array

multiplication operator to create a vector, storing in the new vector the total pay for every employee.

```
>> hours = v(1:2:length(v))
hours =
    33    40    20

>> payrate = v(2:2:length(v))
payrate =
   10.5000   18.0000    7.5000

>> totpay = hours .* payrate
totpay =
  346.5000  720.0000  150.0000
```

41) A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 10 separate times; they are in vector variables *r* and *h*. Find the volume from each trial, which is given by $\Pi r^2 h$. Also use logical indexing first to make sure that all measurements were valid (> 0).

```
>> r = [5.501  5.5  5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
>> h = [11.11 11.1 11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];
>> all(r>0 & h>0)
ans =
     1
>> vol = pi * r.^2 .* h
```

42) For the following matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \qquad B = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

- Give the result of 3*A.

$$\begin{bmatrix} 3 & 12 \\ 9 & 6 \end{bmatrix}$$

- Give the result of A*C.

$$\begin{bmatrix} 19 & 6 & 13 \\ 17 & 8 & 19 \end{bmatrix}$$

- Are there any other matrix multiplications that can be performed? If so, list them.

C*B

43) For the following vectors and matrices A, B, and C:

$A = \begin{bmatrix} 4 & 1 & -1 \\ 2 & 3 & 0 \end{bmatrix}$ $B = \begin{bmatrix} 1 & 4 \end{bmatrix}$ $C = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$

Perform the following operations, if possible. If not, just say it can't be done!

```
A * B
```

No, inner dimensions do not agree

```
B * C
```

14

```
C * B
```

$\begin{bmatrix} 2 & 8 \\ 3 & 12 \end{bmatrix}$

44) The matrix variable *rainmat* stores the total rainfall in inches for some districts for the years 2010-2013. Each row has the rainfall amounts for a given district. For example, if *rainmat* has the value:

```
>> rainmat
ans =
  25   33   29   42
  53   44   40   56
 etc.
```

district 1 had 25 inches in 2010, 33 in 2011, etc. Write expression(s) that will find the number of the district that had the highest total rainfall for the entire four year period.

```
>> rainmat = [25 33 29 42; 53 44 40 56];
>> large = max(max(rainmat))
large =
    56
>> linind = find(rainmat== large)
linind =
     8
```

```
>> floor(linind/4)
ans =
     2
```

45) Generate a vector of 20 random integers, each in the range from 50 to 100. Create a variable *evens* that stores all of the even numbers from the vector, and a variable *odds* that stores the odd numbers.

```
>> nums = randi([50, 100], 1, 20);
>> evens = nums(rem(nums,2)==0);
>> odds = nums(rem(nums,2)~=0);
```

46) Assume that the function **diff** does not exist. Write your own expression(s) to accomplish the same thing for a vector.

```
>> vec = [5 11 2 33 -4]
vec =
     5    11     2    33    -4
>> v1 = vec(2:end);
>> v2 = vec(1:end-1);
>> v1-v2
ans =
     6    -9    31   -37
```

47) 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
```

Chapter 3: Introduction to MATLAB Programming

Exercises

1) Using the top-down design approach, write an algorithm for making a sandwich.

- Get the ingredients

- Get the utensils
- Assemble the sandwich

  Get the ingredients:
  - Get the bread
  - Get the cheese
  - Get the condiments

    Get the bread:
    - Open the bread box
    - Select desired bread
    - Open bag and remove 2 slices

    etc.

2) 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. Use **help** to view the comment in your script.

Ch3Ex2.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)
```

3) Write a statement that prompts the user for his/her favorite number.

```
favnum = input('What is your favorite number: ');
```

4) Write a statement that prompts the user for his/her name.

```
uname = input('What is your name: ', 's');
```

5) 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
```

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) When would you use **disp** instead of **fprintf**? When would you use **fprintf** instead of **disp**?

The **disp** function is used when no formatting is required. It is also easier to print vectors and matrices using disp. The **fprintf** function is used for formatted output.

9) 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)
```

10) 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. 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
```

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: ');

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

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

thirdsideii.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 = side3(b,c,alpha);
fprintf('\nThe third side is %.3f\n', a)
```

side3.m

```
function a = side3(b,c,alpha)
a = sqrt(b^2 + c^2 -2*b*c*cos(alpha));
end
```

11) Write a script that will prompt the user for a character, and will print it twice; once left-justified in a field width of 5, and again right-justified in a field width of 3.

Ch3Ex11.m

```
mych = input('Enter a character: ', 's');
fprintf('Here it is: %-5c and again: %3c\n',mych,mych)
```

12) Write a script *lumin* that will calculate and print the luminosity L of a star in Watts. The luminosity L is given by $L = 4 \pi d^2 b$ where d is the distance from the sun in meters and b is the brightness in Watts/meters^2. Here is an example of executing the script:

```
>> lumin
This script will calculate the luminosity of a star.
When prompted, enter the star's distance from the sun
  in meters, and its brightness in W/meters squared.

Enter the distance: 1.26e12
Enter the brightness: 2e-17
The luminosity of this star is 399007399.75 watts
```

lumin.m

```
% Calculates the luminosity of a star

disp('This script will calculate the luminosity of a star.')
disp('When prompted, enter the star''s distance from the sun')
fprintf('  in meters, and its brightness in W/meters squared.\n\n')

d = input('Enter the distance: ');
b = input('Enter the brightness: ');
L = 4*pi*d^2*b;

fprintf('The luminosity of this star is %.2f watts\n', L)
```

13) A script *iotrace* has been written. Here's what the desired output looks like:

```
>> iotrace
Please enter a number: 33
Please enter a character: x
Your number is 33.00
Your char is      x!
```

Fix this script so that it works as shown above:

iotrace.m

```
mynum = input('Please enter a number:~~\n~~ ');

mychar = input('Please enter a character: ', 's');
```

```
fprintf('Your number is %.2f\n', mynum)

fprintf('Your char is %6c!\n', mychar)
```

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

Ch3Ex14.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+')
```

15) 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

Ch3Ex15.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')
```

16) When would it be important to use **legend** in a plot?

When you have more than one plot in a single Figure Window.

17) Why do we always suppress all assignment statements in scripts?

- So we don't just see the variable = and then the value.

- So we can control exactly what the output looks like.
- So we can format the output.

18) 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
```

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.

Ch3Ex18.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')
```

19) Generate a random integer *n*, create a vector of the integers 1 through *n* in steps of 2, square them, and plot the squares.

Ch3Ex19.m

```
% Create a vector of integers 1:2:n where n is random
% square them and plot the squares

n = randi([1,50])
vec = 1:2:n;
vecsq = vec .^ 2;
plot(vecsq,'k*')
title('Squares of integers')
```

20) Create a *3 x 6* matrix of random integers, each in the range of 50 - 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 of 50 - 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
```

21) A particular 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
```

Ch3Ex21.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')
```

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
```

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.

24) Write a function *mwh_to_gj* that will convert 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
```

25) 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.
- The scope of variables is different: scripts use the base workspace, whereas functions have their own workspaces.

26) In quantum mechanics, the angular wavelength for a wavelength λ is defined as $\lambda/2\pi$. Write a function named *makeitangular* that will receive the wavelength as an input argument, and will return the angular wavelength.

makeitangular.m

```
function angwave = makeitangular(wavelength)
angwave = wavelength/(2*pi);
end
```

27) Write a *fives* function that will receive two arguments for the number of rows and columns, and will return a matrix with that size of all fives.

fives.m

```
function five = fives(r,c)
% Returns a matrix of fives of specified size
% Format of call: fives(rows, cols)
% Returns a rows by cols matrix of all fives

% Initialization
five = zeros(r,c) + 5;


end
```

28) Write a function *isdivby4* that will receive an integer input argument, and will return **logical** 1 for **true** if the input argument is divisible by 4, or **logical false** if it is not.

isdivby4.m

```
function out = isdivby4(inarg)
% Returns 1 for true if the input argument is
%  divisible by 4 or 0 for false if not
% Format of call: isdivby4(input arg)
% Returns whether divisible by 4 or not

out = rem(inarg,4) == 0;
end
```

29) Write a function *isint* that will receive a number input argument *innum*, and will return 1 for **true** if this number is an integer, or 0 for **false** if not. Use the fact that *innum* should be equal to **int32(innum)** if it is an integer. Unfortunately, due to round-off errors, it should be noted that it is possible to get **logical** 1 for **true** if the input argument is close to an integer. Therefore the output may not be what you might expect, as shown here.

```
>> isint(4)
ans =
     1
>> isint(4.9999)
ans =
     0
>> isint(4.999999999999999999999999999)
ans =
     1
```

isint.m

```
function out = isint(innum)
% Returns 1 for true if the argument is an integer
% Format of call: isint(number)
% Returns logical 1 iff number is an integer

out = innum == int32(innum);
end
```

30) A Pythagorean triple is a set of positive integers (a,b,c) such that $a^2 + b^2 = c^2$. Write a function *ispythag* that will receive three positive integers (a, b, c in that order) and will return **logical** 1 for **true** if they form a Pythagorean triple, or 0 for **false** if not.

ispythag.m

```
function out = ispythag(a,b,c)
% Determines whether a, b, c are a Pythagorean
%  triple or not
% Format of call: ispythag(a,b,c)
% Returns logical 1 if a Pythagorean triple
```

```
out = a^2 + b^2 == c^2;
end
```

31) 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
```

32) Write a function 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
```

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!

Ch3Ex33.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) In thermodynamics, the Carnot efficiency is the maximum possible efficiency of a heat engine operating between two reservoirs at different temperatures. The Carnot efficiency is given as

$$\eta = 1 - \frac{T_C}{T_H}$$

where T_C and T_H are the absolute temperatures at the cold and hot reservoirs, respectively. Write a script "carnot" that will prompt the user for the two reservoir temperatures in Kelvin, call a function to calculate the Carnot efficiency, and then print the corresponding Carnot efficiency to 3 decimal places. Also write the function.

carnot.m

```
% Calculates the Carnot efficiency, given the temps
% of cold and hot reservoirs, error-checking both

Tc = input('Enter the cold reservoir temperature: ');
Th = input('Enter the hot reservoir temperature: ');

carnotEff = calcCarnot(Tc, Th);
```

```
fprintf('The Carnot efficiency is %.3f\n',carnotEff)
```

calcCarnot.m

```
function eff = calcCarnot(Tc, Th)
eff = 1 - (Tc/Th);
end
```

35) Many mathematical models in engineering use the exponential function. The general form of the exponential decay function is:

$$y(t) = Ae^{-\tau t}$$

where A is the initial value at t=0, and τ is the time constant for the function. Write a script to study the effect of the time constant. To simplify the equation, set A equal to 1. Prompt the user for two different values for the time constant, and for beginning and ending values for the range of a t vector. Then, calculate two different y vectors using the above equation and the two time constants, and graph both exponential functions on the same graph within the range the user specified. Use a function to calculate y. Make one plot red. Be sure to label the graph and both axes. What happens to the decay rate as the time constant gets larger?

Ch3Ex35.m

```
A = 1;

tau1 = input('Enter a time constant: ');
tau2 = input('Enter another time constant: ');

tstart = input('Enter the beginning t: ');
tend = input('Enter the end of t: ');

t = linspace(tstart,tend);

y1 = expfn(A, t, tau1);
y2 = expfn(A, t, tau2);

plot(t,y1,'r*',t,y2,'go')
xlabel('x')
ylabel('y')
title('Exp function')
legend('tau1','tau2')
```

expfn.m

```
function y = expfn(A,t,tau)
y = A * exp(-tau*t);
end
```

Chapter 4: Selection Statements

Exercises

1) Write a script that tests whether the user can follow instructions. It prompts the user to enter an 'x'. If the user enters anything other than an 'x', it prints an error message – otherwise, the script does nothing.

Ch4Ex1.m

```
% Can the user follow instructions??

inx = input('Enter an x: ', 's');
if inx ~= 'x'
    fprintf('That was no x!\n')
end
```

2) Write a function *nexthour* that receives one integer argument, which is an hour of the day, and returns the next hour. This assumes a 12-hour clock; so, for example, the next hour after 12 would be 1. Here are two examples of calling this function.

```
>> fprintf('The next hour will be %d.\n',nexthour(3))
The next hour will be 4.
>> fprintf('The next hour will be %d.\n',nexthour(12))
The next hour will be 1.
```

nexthour.m

```
function outhour = nexthour(currenthour)
% Receives an integer hour of the day and
% returns the next integer hour
% Format of call: nexthour(hour of day)
% Returns the next integer hour

outhour = currenthour + 1;
if outhour == 13
    outhour = 1;
end
end
```

3) The speed of a sound wave is affected by the temperature of the air. At 0° C, the speed of a sound wave is 331 m/sec. The speed increases

by approximately 0.6 m/sec for every degree (in Celsius) above 0; this is a reasonably accurate approximation for 0 – 50 degrees C. So, our equation for the speed in terms of a temperature C is:

```
speed = 331 + 0.6 * C
```

Write a script "soundtemp" that will prompt the user for a temperature in Celsius in the range from 0 to 50 inclusive, and will calculate and print the speed of sound at that temperature if the user enters a temperature in that range, or an error message if not. Here are some examples of using the script:

```
>> soundtemp
Enter a temp in the range 0 to 50: -5.7
Error in temperature
>> soundtemp
Enter a temp in the range 0 to 50: 10
For a temperature of 10.0, the speed is 337.0
>> help soundtemp
  Calculates and prints the speed of sound given a
  temperature entered by the user
```

soundtemp.m

```
% Calculates and prints the speed of sound given a
% temperature entered by the user
tempC = input('Enter a temp in the range 0 to 50: ');
if tempC < 0 || tempC > 50
    disp('Error in temperature')
else
    speed = 331 + 0.6 * tempC;
    fprintf('For a temperature of %.1f, ', tempC)
    fprintf('the speed is %.1f\n', speed)
end
```

4) When would you use just an **if** statement and not an **if-else**?

In any case in which if the condition is false, no action is required.

5) Come up with "trigger words" in a problem statement that would tell you when it would be appropriate to use **if**, **if-else**, or **switch** statements.

- The word "if" is a big clue!!
- "in the case of"
- "on the other hand"

- "otherwise"

6) Write a statement that will store **logical true** in a variable named "isit" if the value of a variable "x" is in the range from 0 to 10, or **logical false** if not. Do this with just one assignment statement, with no **if** or **if-else** statement!

```
isit = x > 0 && x < 10
```

7) The Pythagorean theorem states that for a right triangle, the relationship between the length of the hypotenuse *c* and the lengths of the other sides *a* and *b* is given by:

$c^2 = a^2 + b^2$

Write a script that will prompt the user for the lengths *a* and *c*, call a function *findb* to calculate and return the length of *b*, and print the result. Note that any values of *a* or *c* that are less than or equal to zero would not make sense, so the script should print an error message if the user enters any invalid value. Here is the function *findb*:

findb.m

```
function b = findb(a,c)
% Calculates b from a and c
b = sqrt(c^2 - a^2);
end
```

Ch4Ex7.m

```
c = input('Enter the length of c: ');
a = input('Enter the length of a: ');
if a > 0 && c > 0
    b = findb(a,c);
    fprintf('The length of b is %.1f\n', b)
else
    fprintf('Error in input\n')
end
```

8) The area A of a rhombus is defined as $A = \frac{d_1 d_2}{2}$, where d_1 and d_2 are the lengths of the two diagonals. Write a script *rhomb* that first prompts the user for the lengths of the two diagonals. If either is a negative number or zero, the script prints an error message. Otherwise, if they are both positive, it calls a function *rhombarea* to return the area of the rhombus, and prints the result. Write the

function, also! The lengths of the diagonals, which you can assume are in inches, are passed to the *rhombarea* function.

rhomb.m

```
% Prompt the user for the two diagonals of a rhombus,
% call a function to calculate the area, and print it

d1 = input('Enter the first diagonal: ');
d2 = input('Enter the second diagonal: ');

if d1 <= 0 || d2 <= 0
    disp('Error in diagonal')
else
    rharea = rhombarea(d1,d2);
    fprintf('The area of the rhombus is %.2f\n', rharea)
end
```

rhombarea.m

```
function rarea = rhombarea(d1,d2)
% Calculates the area of a rhombus given the
% lengths of the two diagonals
% Format of call: rhombarea(diag1, diag2)
% Returns the area of the rhombus

rarea = (d1*d2)/2;
end
```

9) A data file “parttolerance.dat” stores, on one line, a part number, and the minimum and maximum values for the valid range that the part could weigh. Write a script *parttol* that will read these values from the file, prompt the user for a weight, and print whether or not that weight is within range.

For example, IF the file stores the following:

```
>> type parttolerance.dat

123  44.205  44.287
```

Here might be examples of executing the script:

```
>> parttol
Enter the part weight: 44.33
The part 123 is not in range
>> parttol
Enter the part weight: 44.25
The part 123 is within range
```

parttol.m

```
load parttolerance.dat
partno = parttolerance(1);
minwt = parttolerance(2);
maxwt = parttolerance(3);
partwt = input('Enter the part weight: ');
if partwt > minwt && partwt < maxwt
   fprintf('The part %d is within range\n',partno)
else
   fprintf('The part %d is not in range\n',partno)
end
```

10) Write a script that will prompt the user for a character. It will create an x-vector that has 50 numbers, equally spaced between -2**π** and 2**π**, and then a y-vector which is cos(x). If the user entered the character 'r', it will plot these vectors with red *s – otherwise, for any other character it will plot the points with green +s.

Ch4Ex10.m

```
color = input('Enter a char for a color: ', 's');
x = linspace(-2*pi, 2*pi, 50);
y = cos(x);
if color == 'r'
    plot(x,y,'r*')
else
    plot(x,y,'g+')
end
```

11) Simplify this statement:

```
if number > 100
   number = 100;
else
   number = number;
end
```

```
if number > 100
   number = 100;
end
```

12) Simplify this statement:

```
if val >= 10
   disp('Hello')
elseif val < 10
```

```
        disp('Hi')
    end
```

```
if val >= 10
   disp('Hello')
else
   disp('Hi')
end
```

13) The continuity equation in fluid dynamics for steady fluid flow through a stream tube equates the product of the density, velocity, and area at two points that have varying cross-sectional areas. For incompressible flow, the densities are constant so the equation is $A_1V_1 = A_2V_2$. If the areas and V_1 are known, V_2 can be found as $\frac{A_1}{A_2}V_1$. Therefore, whether the velocity at the second point increases or decreases depends on the areas at the two points. Write a script that will prompt the user for the two areas in square feet, and will print whether the velocity at the second point will increase, decrease, or remain the same as at the first point.

Ch4Ex13.m

```
% Prints whether the velocity at a point in a stream tube
% will increase, decrease, or remain the same at a second
% point based on the cross-sectional areas of two points

a1 = input('Enter the area at point 1: ');
a2 = input('Enter the area at point 2: ');

if a1 > a2
    disp('The velocity will increase')
elseif a1 < a2
    disp('The velocity will decrease')
else
    disp('The velocity will remain the same')
end
```

14) Write a function *eqfn* that will calculate f(x) = $x^2 + \frac{1}{x}$ for all elements of x. Since division by 0 is not possible, if any element in x is zero, the function will instead return a flag of -99. Here are examples of using this function:

```
>> vec = [5  0  11  2];
>> eqfn(vec)
```

```
ans =
   -99
>> result = eqfn(4)
result =
   16.2500
>> eqfn(2:5)
ans =
    4.5000    9.3333   16.2500   25.2000
```

```
function fofx = eqfn(x)
if any(any(x==0))
    fofx = -99;
else
    fofx = x.^2 + 1./x;
end
end
```

15) In chemistry, the pH of an aqueous solution is a measure of its acidity. The pH scale ranges from 0 to 14, inclusive. A solution with a pH of 7 is said to be *neutral*, a solution with a pH greater than 7 is *basic*, and a solution with a pH less than 7 is *acidic*. Write a script that will prompt the user for the pH of a solution, and will print whether it is neutral, basic, or acidic. If the user enters an invalid pH, an error message will be printed.

Ch4Ex15.m

```
% Prompts the user for the pH of a solution and prints
% whether it is basic, acidic, or neutral

ph = input('Enter the pH of the solution: ');
if ph >=0 && ph <= 14
    if ph < 7
       disp('It is acidic')
    elseif ph == 7
       disp('It is neutral')
    elseif ph > 7
       disp('It is basic')
    end
else
    disp('Error in pH!')
end
```

16) Write a function *flipvec* that will receive one input argument. If the input argument is a row vector, the function will reverse the order and return a new row vector. If the input argument is a column vector, the

function will reverse the order and return a new column vector. If the input argument is a matrix or a scalar, the
function will return the input argument unchanged.

flipvec.m

```
function out = flipvec(vec)
% Flips it if it's a vector, otherwise
% returns the input argument unchanged
% Format of call: flipvec(vec)
% Returns flipped vector or unchanged

[r c] = size(vec);

if r == 1 && c > 1
    out = fliplr(vec);
elseif c == 1 && r > 1
    out = flipud(vec);
else
    out = vec;
end
end
```

17) In a script, the user is supposed to enter either a ‘y’ or ‘n’ in response to a prompt. The user’s input is read into a character variable called “letter”. The script will print “OK, continuing” if the user enters either a ‘y’ or ‘Y’ or it will print “OK, halting” if the user enters a ‘n’ or ‘N’ or “Error” if the user enters anything else. Put this statement in the script first:

```
letter = input('Enter your answer: ', 's');
```

Write the script using a single nested **if-else** statement (**elseif** clause is permitted).

Ch4Ex17.m

```
% Prompts the user for a 'y' or 'n' answer and responds
% accordingly, using an if-else statement

letter = input('Enter your answer: ', 's');

if letter == 'y' || letter == 'Y'
    disp('OK, continuing')
elseif letter == 'n' || letter == 'N'
    disp('OK, halting')
else
    disp('Error')
end
```

18) Write the script from the previous exercise using a **switch** statement instead.

Ch4Ex18.m

```
% Prompts the user for a 'y' or 'n' answer and responds
% accordingly, using a switch statement

letter = input('Enter your answer: ', 's');

switch letter
    case {'y', 'Y'}
       disp('OK, continuing')
    case {'n', 'N'}
       disp('OK, halting')
    otherwise
       disp('Error')
end
```

19) In aerodynamics, the Mach number is a critical quantity. It is defined as the ratio of the speed of an object (e.g., an aircraft) to the speed of sound. If the Mach number is less than 1, the flow is subsonic; if the Mach number is equal to 1, the flow is transonic; if the Mach number is greater than 1, the flow is supersonic. Write a script that will prompt the user for the speed of an aircraft and the speed of sound at the aircraft's current altitude and will print whether the condition is subsonic, transonic, or supersonic.

Ch4Ex19.m

```
% Prints whether the speed of an object is subsonic,
% transonic, or supersonic based on the Mach number

plane_speed = input('Enter the speed of the aircraft: ');
sound_speed = input('Enter the speed of sound: ');
mach = plane_speed/sound_speed;

if mach < 1
    disp('Subsonic')
elseif mach == 1
    disp('Transonic')
else
    disp('Supersonic')
end
```

20) Write a script that will generate one random integer, and will print whether the random integer is an even or an odd number. (Hint: an

even number is divisible by 2, whereas an odd number is not; so check the remainder after dividing by 2.)

Ch4Ex20.m

```
% Generates a random integer and prints whether it is even %
or odd

ranInt = randi([1, 100]);

if rem(ranInt,2) == 0
    fprintf('%d is even\n', ranInt)
else
    fprintf('%d  is odd\n', ranInt)
end
```

Global temperature changes have resulted in new patterns of storms in many parts of the world. Tracking wind speeds and a variety of categories of storms is important in understanding the ramifications of these temperature variations. Programs that work with storm data will use selection statements to determine the severity of storms and also to make decisions based on the data.

21) Whether a storm is a tropical depression, tropical storm, or hurricane is determined by the average sustained wind speed. In miles per hour, a storm is a tropical depression if the winds are less than 38 mph. It is a tropical storm if the winds are between 39 and 73 mph, and it is a hurricane if the wind speeds are >= 74 mph. Write a script that will prompt the user for the wind speed of the storm, and will print which type of storm it is.

Ch4Ex21.m

```
% Prints whether a storm is a tropical depression, tropical
% storm, or hurricane based on wind speed

wind = input('Enter the wind speed of the storm: ');

if wind < 38
    disp('Tropical depression')
elseif wind >= 38 && wind < 73
    disp('Tropical storm')
else
    disp('Hurricane')
end
```

22) The Beaufort Wind Scale is used to characterize the strength of winds. The scale uses integer values and goes from a force of 0, which is no wind, up to 12, which is a hurricane. The following script first generates a random force value. Then, it prints a message regarding what type of wind that force represents, using a **switch** statement. You are to re-write this **switch** statement as one nested **if-else** statement that accomplishes exactly the same thing. You may use **else** and/or **elseif** clauses.

Ch4Ex22.m

```
ranforce = randi([0, 12]);

switch ranforce
    case 0
        disp('There is no wind')
    case {1,2,3,4,5,6}
        disp('There is a breeze')
    case {7,8,9}
        disp('This is a gale')
    case {10,11}
        disp('It is a storm')
    case 12
        disp('Hello, Hurricane!')
end

if ranforce == 0
    disp('There is no wind')
elseif ranforce >= 1 && ranforce <= 6
    disp('There is a breeze')
elseif ranforce >= 7 && ranforce <= 9
    disp('This is a gale')
elseif ranforce == 10 || ranforce == 11
    disp('It is a storm')
else
    disp('Hello, Hurricane!')
end
```

23) Rewrite the following **switch** statement as one nested **if-else** statement (**elseif** clauses may be used). Assume that there is a variable *letter* and that it has been initialized.

```
switch letter
     case 'x'
          disp('Hello')
     case {'y', 'Y'}
          disp('Yes')
```

```
          case 'Q'
               disp('Quit')
          otherwise
               disp('Error')
     end
```

Ch4Ex23.m

```
letter = char(randi([97,122]));

if letter == 'x'
    disp('Hello')
elseif letter == 'y' || letter == 'Y'
    disp('Yes')
elseif letter == 'Q'
    disp('Quit')
else
    disp('Error')
end
```

24) Rewrite the following nested **if-else** statement as a **switch** statement that accomplishes exactly the same thing. Assume that *num* is an integer variable that has been initialized, and that there are functions *f1*, *f2*, *f3*, and *f4*. Do not use any **if** or **if-else** statements in the actions in the **switch** statement, only calls to the four functions.

```
     if num < -2 || num > 4
          f1(num)
     else
          if num <= 2
               if num >= 0
                    f2(num)
               else
                    f3(num)
               end
          else
               f4(num)
          end
     end
```

Ch4Ex24.m

```
% To run this, would need to create functions

switch num
    case {-2, -1}
        f3(num)
    case {0, 1, 2}
```

```
        f2(num)
    case {3, 4}
        f4(num)
    otherwise
        f1(num)
end
```

25) Write a script *areaMenu* that will print a list consisting of "cylinder", "circle", and "rectangle". It prompts the user to choose one, and then prompts the user for the appropriate quantities (e.g., the radius of the circle) and then prints its area. If the user enters an invalid choice, the script simply prints an error message. The script should use a nested **if-else** statement to accomplish this. Here are two examples of running it (units are assumed to be inches).

```
>> areaMenu
Menu
1. Cylinder
2. Circle
3. Rectangle
Please choose one: 2
Enter the radius of the circle: 4.1
The area is 52.81

>> areaMenu
Menu
1. Cylinder
2. Circle
3. Rectangle
Please choose one: 3
Enter the length: 4
Enter the width: 6
The area is 24.00
```

areaMenu.m

```
% Prints a menu and calculates area of user's choice

disp('Menu')
disp('1. Cylinder')
disp('2. Circle')
disp('3. Rectangle')
sh = input('Please choose one: ');
if sh == 1
    rad = input('Enter the radius of the cylinder: ');
    ht = input('Enter the height of the cylinder: ');
    fprintf('The surface area is %.2f\n', 2*pi*rad*ht)
elseif sh == 2
```

```
    rad = input('Enter the radius of the circle: ');
    fprintf('The area is %.2f\n', pi*rad*rad)
elseif sh == 3
    len = input('Enter the length: ');
    wid = input('Enter the width: ');
    fprintf('The area is %.2f\n', len*wid)
else
    disp('Error!  Not a valid choice.')
end
```

26) Modify *the areaMenu* script to use a **switch** statement to decide which area to calculate.

areaMenuSwitch.m

```
% Prints a menu and calculates area of user's choice

disp('Menu')
disp('1. Cylinder')
disp('2. Circle')
disp('3. Rectangle')
sh = input('Please choose one: ');
switch sh
    case 1
        rad = input('Enter the radius of the cylinder: ');
        ht = input('Enter the height of the cylinder: ');
        fprintf('The surface area is %.2f\n', 2*pi*rad*ht)
    case 2
        rad = input('Enter the radius of the circle: ');
        fprintf('The area is %.2f\n', pi*rad*rad)
    case 3
        len = input('Enter the length: ');
        wid = input('Enter the width: ');
        fprintf('The area is %.2f\n', len*wid)
    otherwise
        disp('Error!  Not a valid choice.')
end
```

27) Write a script that will prompt the user for a string and then print whether it was empty or not.

Ch4Ex27.m

```
instring = input('Enter a string: ', 's');
if isempty(instring)
    fprintf('Not much of a string there!\n')
else
    fprintf('Thanks!\n')
```

```
end
```

28) Simplify this statement:

```
if iskeyword('else') == 1    % == 1 is redundant
    disp('Cannot use as a variable name')
end
```

29) Store a value in a variable and then use **isa** to test to see whether or not it is the type **double**.

```
>> val = 43;
>> isa(val, 'double')
ans =
     1
```

30) Write a function called “makemat” that will receive two row vectors as input arguments, and from them create and return a matrix with two rows. You may not assume that the length of the vectors is known. Also, the vectors may be of different lengths. If that is the case, add 0’s to the end of one vector first to make it as long as the other. For example, a call to the function might be:

```
>>makemat(1:4, 2:7)
ans =

     1     2     3     4     0     0
     2     3     4     5     6     7
```

makemat.m

```
function outmat = makemat(v1,v2)

len1 = length(v1);
len2 = length(v2);

if len1 ~= len2
   if len1>len2
      diff = len1-len2;
      addend = zeros(1,diff);
      v2 = [v2 addend];
   else
      diff = len2-len1;
      addend = zeros(1,diff);
      v1 = [v1 addend];
   end
end
```

```
outmat = [v1;v2];
```

Chapter 5: Loop Statements and Vectorizing Code

Exercises

1) Write a **for** loop that will print the column of real numbers from 2.7 to 3.5 in steps of 0.2.

Ch5Ex1.m

```
for i = 2.7: 0.2: 3.5
    disp(i)
end
```

2) In the Command Window, write a **for** loop that will iterate through the integers from 32 to 255. For each, show the corresponding character from the character encoding. Play with this! Try printing characters beyond the standard ASCII, in small groups. For example, print the characters that correspond to integers from 300 to 340.

```
>> for i = 32:255
    disp(char(i))
end

!
"
#
$
%

etc.
```

3) Prompt the user for an integer *n* and print "I love this stuff!" n times.

Ch5Ex3.m

```
% Prompts the user for an integer n and prints
% "I love this stuff" n times

n = input('Enter an integer: ');
for i = 1:n
   disp('I love this stuff!')
end
```

4) When would it matter if a **for** loop contained `for i = 1:4` vs. `for i = [3 5 2 6]`, and when would it not matter?

It would matter if the value of the loop variable was being used in the action of the loop. It would not matter if the loop variable was just being used to count how many times to execute the action of the loop.

5) Write a function *sumsteps2* that calculates and returns the sum of 1 to *n* in steps of 2, where *n* is an argument passed to the function. For example, if 11 is passed, it will return 1 + 3 + 5 + 7 + 9 + 11. Do this using a **for** loop. Calling the function will look like this:

```
>> sumsteps2(11)
ans =
    36
```

sumsteps2.m

```
function outsum = sumsteps2(n)
% sum from 1 to n in steps of 2
% Format of call: sumsteps2(n)
% Returns 1 + 3 + ... + n

outsum = 0;
for i = 1:2:n
    outsum = outsum + i;
end
end
```

6) Write a function *prodby2* that will receive a value of a positive integer *n* and will calculate and return the product of the odd integers from 1 to *n* (or from 1 to *n-1* if *n* is even). Use a **for** loop.

prodby2.m

```
function out = prodby2(n)
% Calculates and returns 1*3*5*..*n
% Format of call: prodby2(n)
% Returns product from 1 to n in steps of 2

out = 1;
for i = 1:2:n
    out = out * i;
end
end
```

7) Write a script that will:

- generate a random integer in the inclusive range from 2 to 5

- loop that many times to
    - prompt the user for a number
    - print the sum of the numbers entered so far with one decimal place

Ch5Ex7.m

```
% Generate a random integer and loop to prompt the
% user for that many numbers and print the running sums

ranint = randi([2,5]);
runsum = 0;
for i = 1:ranint
    num = input('Please enter a number: ');
    runsum = runsum + num;
    fprintf('The sum so far is %.1f\n', runsum)
end
```

8) Write a script that will load data from a file into a matrix. Create the data file first, and make sure that there is the same number of values on every line in the file so that it can be loaded into a matrix. Using a **for** loop, it will then create a subplot for every row in the matrix, and will plot the numbers from each row element in the Figure Window.

xfile.dat

```
4     9    22
30    18    4
```

Ch5Ex8.m

```
% load data from a file and plot data
% from each line in a separate plot in a subplot

load xfile.dat
[r c] = size(xfile);
for i = 1:r
    subplot(1,r,i)
    plot(xfile(i,:),'k*')
end
```

9) Write code that will prompt the user for 4 numbers, and store them in a vector. Make sure that you preallocate the vector!

```
vec = zeros(1,4);
for i = 1:4
     vec(i) = input('Enter a number: ');
end
```

10) Write a **for** loop that will print the elements from a vector variable in sentence format, regardless of the length of the vector. For example, if this is the vector:

```
>> vec = [5.5 11 3.45];
```

this would be the result:

```
Element 1 is 5.50.
Element 2 is 11.00.
Element 3 is 3.45.
```

The **for** loop should work regardless of how many elements are in the vector.

```
>> vec = [44 11 2 9 6];
>> for i = 1:length(vec)
      fprintf('Element %d is %.2f\n',i,vec(i))
   end
Element 1 is 44.00
Element 2 is 11.00
Element 3 is 2.00
Element 4 is 9.00
Element 5 is 6.00
```

11) Execute this script and be amazed by the results! You can try more points to get a clearer picture, but it may take a while to run.

Ch5Ex11.m

```
clear
clf
x = rand;
y = rand;
plot(x,y)

hold on
for it = 1:10000
   choic = round(rand*2);
   if choic == 0
      x = x/2;
      y = y/2;
   elseif choic == 1
      x = (x+1)/2;
      y = y/2;
   else
      x = (x+0.5)/2;
      y = (y+1)/2;
   end
   plot(x,y,'r*')
```

```
   hold on
end
```

12) A machine cuts N pieces of a pipe. After each cut, each piece of pipe is weighed and its length is measured; these 2 values are then stored in a file called *pipe.dat* (first the weight and then the length on each line of the file). Ignoring units, the weight is supposed to be between 2.1 and 2.3, inclusive, and the length is supposed to be between 10.3 and 10.4, inclusive. The following is just the beginning of what will be a long script to work with these data. For now, the script will just count how many rejects there are. A reject is any piece of pipe that has an invalid weight and/or length. For a simple example, if N is 3 (meaning three lines in the file) and the file stores:

```
2.14  10.30
2.32  10.36
2.20  10.35
```

there is only one reject, the second one, as it weighs too much. The script would print:

There were 1 rejects.

Ch5Ex12.m

```
% Counts pipe rejects.  Ignoring units, each pipe should be
% between 2.1 and 2.3 in weight and between 10.3 and 10.4
% in length

% read the pipe data and separate into vectors
load pipe.dat
weights = pipe(:,1);
lengths = pipe(:,2);
N = length(weights);

% the programming method of counting
count = 0;

for i=1:N
    if weights(i) < 2.1 || weights(i) > 2.3 || ...
            lengths(i) < 10.3 || lengths(i) > 10.4
        count = count + 1;
    end
end

fprintf('There were %d rejects.\n', count)
```

13) Come up with "trigger" words in a problem statement that would tell you when it's appropriate to use **for** loops and/or nested **for** loops.

"Loop to …", "Iterate through the values…", "Repeat this for values…", "Every time you do this, repeat the process of …" (nested)

14) With a matrix, when would:

- your outer loop be over the rows
- your outer loop be over the columns
- it not matter which is the outer and which is the inner loop?

The outer loop must be over the rows if you want to perform an action for every row; it must be over the columns if you want to perform an action for every column. It does not matter if you simply need to refer to every element in the matrix.

15) Write a function *myones* that will receive two input arguments *n* and *m* and will return an *nxm* matrix of all ones. Do NOT use any built-in functions (so, yes, the code will be inefficient). Here is an example of calling the function:

myones.m

```
function outmat = myones(n,m)
for i = 1:n
    for j = 1:m
        outmat(i,j) = 1;
    end
end
end
```

16) Write a script that will print the following multiplication table:

```
 1
 2  4
 3  6  9
 4  8  12  16
 5  10  15  20  25
```

Ch5Ex16.m

```
% Prints a multiplication table

rows = 5;

for i = 1:rows
    for j = 1:i
```

```
        fprintf('%d  ', i*j)
    end
    fprintf('\n')
end
```

17) Write a function that will receive a matrix as an input argument, and will calculate and return the overall average of all numbers in the matrix. Use loops, not built-in functions, to calculate the average.

matave.m

```
function outave = matave(mat)
% Calculates the overall average of numbers in a matrix
% using the programming methods
% Format of call: matave(matrix)
% Returns the average of all elements

mysum = 0;
[r c] = size(mat);
for i = 1:r
    for j = 1:c
        mysum = mysum + mat(i,j);
    end
end
outave = mysum/(r*c);
end
```

18) Write an algorithm for an ATM program. Think about where there would be selection statements, menus, loops (counted vs. conditional), etc. – but – don't write MATLAB code, just an algorithm (pseudo-code).

```
Read card
Look up account
If invalid,
   print error message and end
else
   Display menu of language choices
   Select language based on button push
   Ask for PIN
   while PIN is invalid (up to 3 tries)
      Ask for PIN again
   end
   if PIN still invalid
      print an error message and end
   else
      carry on!  etc.
```

19) Trace this to figure out what the result will be, and then type it into MATLAB to verify the results.

Ch5Ex19.m

```
count = 0;
number = 8;
while number > 3
     fprintf('number is %d\n', number)
     number = number - 2;
     count = count + 1;
end
fprintf('count is %d\n', count)
```

```
number is 8
number is 6
number is 4
count is 3
```

20) The inverse of the mathematical constant *e* can be approximated as follows:

$$\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n$$

Write a script that will loop through values of *n* until the difference between the approximation and the actual value is less than 0.0001. The script should then print out the built-in value of e^{-1} and the approximation to 4 decimal places, and also print the value of *n* required for such accuracy.

Ch5Ex20.m

```
% Approximates 1/e as (1-1/n)^n, and determines
% the value of n required for accuracy to 4 dec. places

actual = 1 / exp(1);
diff = 1;
n = 0;

while diff >= 0.0001
    n = n + 1;
    approx = (1 - 1/n)^n;
    diff = actual - approx;
end

fprintf('The built-in value of 1/e is %.4f\n',actual)
fprintf('The approximation is %.4f\n',approx)
fprintf('The value of n is %d\n',n)
```

21) Write a script that will generate random integers in the range from 0 to 50, and print them, until one is finally generated that is greater than 25. The script should print how many attempts it took.

Ch5Ex21.m

```
rani = randi([0 50]);
fprintf('The integer is %d\n', rani)
count = 1;
while rani <= 25
    rani = randi([0 50]);
    fprintf('The integer is %d\n', rani)
    count = count + 1;
end
fprintf('Yay, a %d! It took %d tries\n', rani, count)
```

22) Write a script will prompt the user for a keyword in MATLAB, error-checking until a keyword is entered.

Ch5Ex22.m

```
keyw = input('Please enter a MATLAB keyword: ', 's');
while ~iskeyword(keyw)
  keyw = input('Seriously, enter a MATLAB keyword: ', 's');
end
fprintf('Indeed, %s is a keyword!\n', keyw)
```

23) A blizzard is a massive snowstorm. Definitions vary, but for our purposes we will assume that a blizzard is characterized by both winds of 30 mph or higher and blowing snow that leads to visibility of 0.5 miles or less, sustained for at least four hours. Data from a storm one day has been stored in a file *stormtrack.dat*. There are 24 lines in the file, one for each hour of the day. Each line in the file has the wind speed and visibility at a location. Create a sample data file. Read this data from the file and determine whether blizzard conditions were met during this day or not.

Ch5Ex23.m

```
% Reads wind and visibility data hourly from a file and
% determines whether or not blizzard conditions were met

load stormtrack.dat

winds = stormtrack(:,1);
visibs = stormtrack(:,2);
```

```
len = length(winds);
count = 0;
i = 0;
% Loop until blizzard condition found or all data
% has been read

while count < 4 && i < len
   i = i + 1;
   if winds(i) >= 30 && visibs(i) <= .5
       count = count + 1;
   else
       count = 0;
   end
end
if count == 4
    fprintf('Blizzard conditions met\n')
else
    fprintf('No blizzard this time!\n')
end
```

24) Given the following loop:

```
while x < 10
     action
end
```

- For what values of the variable *x* would the action of the loop be skipped entirely?

The action would be skipped entirely if x is greater than or equal to 10 to begin with.

- If the variable *x* is initialized to have the value of 5 before the loop, what would the action have to include in order for this to not be an infinite loop?

The action would have to increment the value of x, so that eventually it becomes greater than or equal to 10.

25) Write a script called *prtemps* that will prompt the user for a maximum Celsius value in the range from -16 to 20; error-check to make sure it's in that range. Then, print a table showing degrees Fahrenheit and degrees Celsius until this maximum is reached. The first value that exceeds the maximum should not be printed. The table should start at 0 degrees Fahrenheit, and increment by 5 degrees Fahrenheit until the max (in Celsius) is reached. Both temperatures

should be printed with a field width of 6 and one decimal place. The formula is C = 5/9 (F-32).

prtemps.m

```
% Prompt for a maximum C temperature and print a table
% showing degrees C and degrees F

fprintf('When prompted, enter a temp in degrees C in')
fprintf(' range -16\n to 20.\n')
maxtemp = input('Enter a maximum temp: ');

% Error-check
while maxtemp < -16 || maxtemp > 20
    maxtemp = input('Error! Enter a maximum temp: ');
end

% Print table include headers
fprintf('    F       C\n');

f = 0;
c = 5/9*(f-32);

while (c <= maxtemp)
   fprintf('%6.1f  %6.1f\n',f,c)
   f = f + 5;
   c = 5/9*(f-32);
end
```

26) Vectorize this code! Write *one* assignment statement that will accomplish exactly the same thing as the given code (assume that the variable *vec* has been initialized):

```
result = 0;
for i = 1:length(vec)
    result = result + vec(i);
end
```

```
>> result = sum(vec)
```

27) Vectorize this code! Write *one* assignment statement that will accomplish exactly the same thing as the given code (assume that the variable *vec* has been initialized):

```
newv = zeros(size(vec));
myprod = 1;
```

```
      for i = 1:length(vec)
         myprod = myprod * vec(i);
         newv(i) = myprod;
      end
      newv % Note: this is just to display the value

>> newv = cumprod(vec)
```

28) The following code was written by somebody who does not know how to use MATLAB efficiently. Rewrite this as a single statement that will accomplish exactly the same thing for a matrix variable *mat* (e.g., vectorize this code):

```
      [r c] = size(mat);
      for i = 1:r
          for j = 1:c
              mat(i,j) = mat(i,j) * 2;
          end
      end

>> mat = mat * 2
```

29) Vectorize the following code. Write one assignment statement that would accomplish the same thing. Assume that *mat* is a matrix variable that has been initialized.

```
[r,c] = size(mat);
val = mat(1,1);
for i = 1:r
    for j = 1:c
        if mat(i,j) < val
            val = mat(i,j);
        end
    end
end
val  % just for display


      val = min(min(mat))
```

30) Vectorize the following code. Write statement(s) that accomplish the same thing, eliminating the loop. Assume that there is a vector v that has a negative number in it, e.g:

```
      >> v = [4 11 22 5 33 -8 3 99 52];

newv = [];
i = 1;
```

```
while v(i) >= 0
   newv(i) = v(i);
   i = i + 1;
end
newv  % Note: just to display
```

```
where = find(v < 0);

newv = v(1:where-1)
```

31) Give some examples of when you would need to use a counted loop in MATLAB, and when you would not.

When you need to prompt the user for values a specified number of times

Using subplot to vary a plot (e.g., sin(x), sin(2x), sin(3x), etc.)

When NOT to use: when performing any operation or function on an array (vector or matrix)

32) For each of the following, decide whether you would use a for loop, a **while** loop, a nested loop (and if so what kind, e.g. a **for** loop inside of another **for** loop, a **while** loop inside of a **for** loop, etc.), or no loop at all. DO NOT WRITE THE ACTUAL CODE.

- sum the integers 1 through 50: **No loop**
- add 3 to all numbers in a vector: **No loop**
- prompt the user for a string, and keep doing this until the string that the user enters is a keyword in MATLAB: **while loop**
- find the minimum in every column of a matrix: **No loop**
- prompt the user for 5 numbers and find their sum: **for loop**
- prompt the user for 10 numbers, find the average and also find how many of the numbers were greater than the average: **for loop**
- generate a random integer n in the range from 10 to 20. Prompt the user for n positive numbers, error-checking to make sure you

get n positive numbers (and just echo print each one): **while loop in a for loop**

- prompt the user for positive numbers until the user enters a negative number. Calculate and print the average of the positive numbers, or an error message if none are entered: **while loop**

33) Write a script that will prompt the user for a quiz grade and error-check until the user enters a valid quiz grade. The script will then echo print the grade. For this case, valid grades are in the range from 0 to 10 in steps of 0.5. Do this by creating a vector of valid grades and then use **any** or **all** in the condition in the **while** loop.

Ch5Ex33.m

```
% Prompt the user for a quiz grade and error-check
% until a valid grade is entered; using any or all

quiz = input('Enter a quiz grade: ');

validgrades = 0: 0.5: 10;

while ~any(quiz == validgrades)
    quiz = input('Invalid! Enter a quiz grade: ');
end

fprintf('The grade is %.1f\n', quiz)
```

34) Which is faster: using **false** or using **logical**(0) to preallocate a matrix to all **logical** zeros? Write a script to test this.

Ch5Ex34.m

```
% Test to see whether logical(0) or false is faster

fprintf('Start with logical(0)\n\n')
tic
mat(1:100, 1:100) = logical(0);
toc

fprintf('Now for false \n\n')
tic
mat2(1:100, 1:100) = false;
toc
```

35) Which is faster: using a **switch** statement or using a nested **if-else**? Write a script to test this.

Ch5Ex35.m

```
% Test which is faster: switch or nested if-else

fprintf('First for if-else\n\n')

tic
for i = 1:1000
    for j = 1:4
        if j == 1
            result = 11;
        elseif j == 2
            result = 22;
        elseif j == 3
            result = 46;
        else
            result = 6;
        end
    end
end
toc

fprintf('Now switch\n\n')

tic
for j = 1:1000
    for k = 1:4
        switch k
            case 1
                res = 11;
            case 2
                res = 22;
            case 3
                res = 46;
            otherwise
                res = 6;
        end
    end
end
toc
```

36) Write a script *beautyofmath* that produces the following output. The script should iterate from 1 to 9 to produce the expressions on the left, perform the specified operation to get the results shown on the right, and print exactly in the format shown here.

```
>> beautyofmath
1 x 8 + 1 = 9
```

```
 12 x 8 + 2 = 98
 123 x 8 + 3 = 987
 1234 x 8 + 4 = 9876
 12345 x 8 + 5 = 98765
 123456 x 8 + 6 = 987654
 1234567 x 8 + 7 = 9876543
 12345678 x 8 + 8 = 98765432
 123456789 x 8 + 9 = 987654321
```

beautyofmath.m

```
% Shows the beauty of math!

leftnum = 0;
for i = 1:9
    leftnum = leftnum * 10 + i;
    result = leftnum * 8 + i;
    fprintf('%d x 8 + %d = %d\n', leftnum, i, result)
end
```

37) 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 WCF is

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

Write a function to receive the temperature and wind speed as input arguments, and return the WCF. Using loops, print a table showing wind chill factors for temperatures ranging from -20 to 55 in steps of 5, and wind speeds ranging from 0 to 55 in steps of 5. Call the function to calculate each wind chill factor.

Ch5Ex37.m

```
% Print table of wind chill factors

% Print column headers
fprintf('%45s\n   ', 'Wind Speeds')
for v = 0:5:55
    fprintf('%7d', v)
end
fprintf('\nTemp\n')

for t = -20:5:55
    fprintf('%3d', t)
    for v = 0:5:55
        fprintf('%7.1f',wcf(t,v))
    end
    fprintf('\n')
```

```
end
```

wcf.m

```
function outwc = wcf(t, v)
% Calculates the wind chill factor
% Format of call: wcf(temperature, wind speed)
% Returns 35.74 + 0.6215T - 35.75(V^0.16) + 0.4275T(V^0.16)

outwc = 35.74 + 0.6215 .* t - 35.75 .* (v.^0.16) + ...
    0.4275 .* t .* (v.^0.16);
end
```

38) Instead of printing the WCFs in the previous problem, create a matrix of WCFs and write them to a file. Use the programming method, using nested loops.

Ch5Ex38.m

```
% Print table of wind chill factors for temperatures
% ranging from -20 to 55F and wind speeds from 0 to 55mph

for t = -4:11
    for v = 0:11
        wcfmat(t+5,v+1) = wcf(5*t,5*v);
    end
end

save wcfs.dat wcfmat -ascii
```

39) Write a script that will prompt the user for N integers, and then write the positive numbers (>= 0) to an ASCII file called *pos.dat* and the negative numbers to an ASCII file called *neg.dat*. Error-check to make sure that the user enters N integers.

Ch5Ex39.m

```
% Prompt the user for N integers, writing the positive
% integers to one file and the negative integers to another

% initialize vectors to store pos and neg integers
posints = [];
negints = [];
% loop n times
n=10;
for i=1:n
    inputnum = input('Enter an integer: ');
    num2 = int32(inputnum);
```

```
    % error check to make sure integers are entered
    while num2 ~= inputnum
        inputnum = input('Invalid! Enter an integer: ');
        num2 = int32(inputnum);
    end
    % add to appropriate vector
    if inputnum < 0
        negints = [negints inputnum];
    else
        posints = [posints inputnum];
    end
end

% write vectors to files
save pos.dat posints -ascii
save neg.dat negints -ascii
```

40) Write a script to add two 30-digit numbers and print the result. This is not as easy as it might sound at first, because integer types may not be able to store a value this large. One way to handle large integers is to store them in vectors, where each element in the vector stores a digit of the integer. Your script should initialize two 30-digit integers, storing each in a vector, and then add these integers, also storing the result in a vector. Create the original numbers using the **randi** function. Hint: add 2 numbers on paper first, and pay attention to what you do!

Ch5Ex40.m

```
num1 = randi([0,9],1,10);
num2 = randi([0,9],1,10);
num1plusnum2 = zeros(1,11);
carry = 0;
for i = 10:-1:1
    sumit = num1(i) + num2(i) + carry;
    if sumit <= 9
       num1plusnum2(i+1) = sumit;
       carry = 0;
    else
       num1plusnum2(i+1) = rem(sumit,10);
       carry = 1;
    end
end
num1plusnum2(1) = carry;

fprintf('   %s\n',num2str(num1))
fprintf('   %s\n', num2str(num2))
```

```
fprintf('%s\n', num2str(num1plusnum2))
```

41) Write a “Guess My Number Game” program. The program generates a random integer in a specified range, and the user (the player) has to guess the number. The program allows the use to play as many times as he/she would like; at the conclusion of each game, the program asks whether the player wants to play again.
The basic algorithm is:
1. The program starts by printing instructions on the screen.
2. For every game:

- the program generates a new random integer in the range from MIN to MAX. Treat MIN and MAX like constants; start by initializing them to 1 and 100
- loop to prompt the player for a guess until the player correctly guesses the integer
    - for each guess, the program prints whether the player’s guess was too low, too high, or correct
- at the conclusion (when the integer has been guessed):
    - print the total number of guesses for that game
    - print a message regarding how well the player did in that game (e.g the player took way too long to guess the number, the player was awesome, etc.). To do this, you will have to decide on ranges for your messages and give a rationale for your decision in a comment in the program.

3. After all games have been played, print a summary showing the average number of guesses.

Ch5Ex41.m

```
% Guess My Number Game

disp('This is a Guess My Number Game!')
disp('I will pick a random integer; see if you can guess it')

MIN = 1;
MAX = 100;

playagain = true;
tot_no_guesses = 0;
count_games = 0;

while playagain
    count_games = count_games + 1;
    guesses = 1;
    randomint = randi([MIN, MAX]);
    yourint = input('Enter your guess: ');
```

```
    while randomint ~= yourint
        if yourint < randomint
            disp('Too low')
            yourint = input('Enter your guess: ');
            guesses = guesses + 1;
        elseif yourint > randomint
            disp('Too high')
            yourint = input('Enter your guess: ');
            guesses = guesses + 1;
        else
            disp('Correct!')
        end
    end
    fprintf('It took you %d guesses, which is ', guesses)
    if guesses < 5
        fprintf(' awesome\n')
    elseif guesses > 10
        fprintf(' pretty weak\n')
    else
        fprintf(' OK\n')
    end
    tot_no_guesses = tot_no_guesses + guesses;
    playagain = upper(input('Play again? ', 's')) == 'Y';
end
fprintf('The ave # of guesses was %.2f\n',
tot_no_guesses/count_games)
```

42) A CD changer allows you to load more than one CD. Many of these have random buttons, which allow you to play random tracks from a specified CD, or play random tracks from random CDs. You are to simulate a play list from such a CD changer using the **randi** function. The CD changer that we are going to simulate can load 3 different CDs. You are to assume that three CDs have been loaded. To begin with, the program should “decide” how many tracks there are on each of the three CDs, by generating random integers in the range from MIN to MAX. You decide on the values of MIN and MAX (look at some CDs; how many tracks do they have? What’s a reasonable range?). The program will print the number of tracks on each CD. Next, the program will ask the user for his or her favorite track; the user must specify which track and which CD it’s on. Next, the program will generate a “playlist” of the N random tracks that it will play, where N is an integer. For each of the N songs, the program will first randomly pick one of the 3 CDs, and then randomly pick one of the tracks from that CD. Finally, the program will print whether the user’s favorite track was played or

not. The output from the program will look something like this depending on the random integers generated and the user's input:

```
There are 15 tracks on CD 1.
There are 22 tracks on CD 2.
There are 13 tracks on CD 3.

What's your favorite track?
Please enter the number of the CD:  4
Sorry, that's not a valid CD.
Please enter the number of the CD: 1
Please enter the track number: 17
Sorry, that's not a valid track on CD 1.
Please enter the track number: -5
Sorry, that's not a valid track on CD 1.
Please enter the track number: 11

Play List:
CD 2 Track 20
CD 3 Track 11
CD 3 Track 8
CD 2 Track 1
CD 1 Track 7
CD 3 Track 8
CD 1 Track 3
CD 1 Track 15
CD 3 Track 12
CD 1 Track 6

Sorry, your favorite track was not played.
```

Ch5Ex42.m

```
MIN = 7;
MAX = 14;
noCDs = 3;
N = 5;

cdtracks = randi([MIN, MAX], 1,3);

for i = 1:noCDs
    fprintf('There are %d tracks on CD %d\n', cdtracks(i), i)
end

fprintf('\nWhat''s your favorite track?\n')
favcd = input('Please enter the number of the CD: ');
while favcd < 1 || favcd > noCDs
    fprintf('Sorry, that''s not a valid CD.\n')
```

```
    favcd = input('Please enter the number of the CD: ');
end
favtrack = input('Please enter the track number: ');
while favtrack < 1 || favtrack > cdtracks(favcd)
    fprintf('Sorry, that''s not a valid track on CD %d\n', ...
       favcd)
    favtrack = input('Please enter the track number: ');
end

played = false;
disp('Play List: ')

for i = 1:N
    rancd = randi([1, noCDs]);
    rantrack = randi([1, cdtracks(rancd)]);
    fprintf('CD %d Track %d\n', rancd, rantrack)
    played = rancd == favcd && rantrack == favtrack;
end

if played
    disp('Your favorite track was played!!')
else
    disp('Your favorite track was not played.')
end
```

43) Write your own code to perform matrix multiplication. Recall that to multiply two matrices, the inner dimensions must be the same.

$$[A]_{m \times n}\,[B]_{n \times p} = [C]_{m \times p}$$

Every element in the resulting matrix C is obtained by:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}.$$

So, three nested loops are required.

mymatmult.m

```
function C = mymatmult(A,B)
% mymatmult performs matrix multiplication
% It returns an empty vector if the matrix
%  multiplication cannot be performed
% Format: mymatmult(matA, matB)

[m, n] = size(A);
[nb, p] = size(B);
```

```
if n ~= nb
    C = [];
else
    % Preallocate C
    C = zeros(m,p);
    % Outer 2 loops iterate through the elements in C
    %   which has dimensions m by p
    for i=1:m
        for j = 1:p
            % Inner loop performs the sum for each
            %  element in C
            mysum = 0;
            for k = 1:n
                mysum = mysum + A(i,k) * B(k,j);
            end
            C(i,j) = mysum;
        end
    end
end
```

Chapter 6: MATLAB Programs

Exercises

1) Given the following function header:

```
function [x, y] = calcem(a, b, c)
```

Which of the following function calls would be valid – and why?

```
[num, val] = calcem(4, 11, 2)
```

VALID - everything matches up (function name, # of input and output arguments)

```
result = calcem(3, 5, 7)
```

VALID – but – one output argument will be lost

2) Write a function that will receive as an input argument a number of kilometers (K). The function will convert the kilometers to miles and to U.S. nautical miles, and return both results. The conversions are: 1K = 0.621 miles and 1 US nautical mile = 1.852 K.

kToMilesNaut.m

```
function [miles, nautmiles] = kToMilesNaut(kms)
% Converts a distance in kilometers to miles and U.S. nautical miles
% Format kToMilesNaut(kilometers)
% Returns miles and then nautical miles

 miles = kms .* 0.621;
 nautmiles = kms ./ 1.852;
 end
```

3) Write a function "splitem" that will receive one vector of numbers as an input argument, and will return two vectors: one with the positive (>=0) numbers from the original vector, and the second the negative numbers from the original vector. Use vectorized code (no loops) in your function.

splitem.m

```
function [pos, neg] = splitem(vec)
pos = vec(find(vec >= 0));
neg = vec(find(vec < 0));
end
```

4) Write a function to calculate the volume and surface area of a hollow cylinder. It receives as input arguments the radius of the cylinder base and the height of the cylinder. The volume is given by $\pi r^2 h$, and the surface area is $2 \pi r h$.

vol_surfarea.m

```
function [vol, surfarea] = vol_surfarea(rad, ht)
% Calculates the volume and surface area of a
% hollow cylinder, given the radius and height
% Format of call: vol_surfarea(radius, height)
% Returns volume then surface area

vol = pi * rad^2 * ht;
surfarea = 2 * pi * rad * ht;
end
```

Satellite navigation systems have become ubiquitous. Navigation systems based in space such as the Global Positioning System (GPS) can send data to handheld personal devices. The coordinate systems that are used to represent locations present this data in several formats.

5) The geographic coordinate system is used to represent any location on Earth as a combination of latitude and longitude values. These

values are angles that can be written in the decimal degrees (DD) form or the degrees, minutes, seconds (DMS) form just like time. For example, 24.5° is equivalent to 24°30’0”. Write a script that will prompt the user for an angle in DD form and will print in sentence format the same angle in DMS form. The script should error-check for invalid user input. The angle conversion is to be done by calling a separate function in the script.

DMSscript.m

```
% Prompts the user for an angle in decimal degrees (DD) form,
% calls a function to converto to degrees, minutes, seconds (DMS)
% and prints the result

angles = input('Enter an angle in decimal degrees form: ');

while angles <= 0
  angles = input('Invalid! Enter an angle in decimal degrees form: ');
end

[d m s] = DMS(angles);

fprintf('%.2f degrees is equivalent to %d deg, %d minutes,\n', ...
    angles, d, m)
fprintf('%.2f seconds\n', s)
```

DMS.m

```
function [degree, minutes, seconds] = DMS(angles)
% converts angle in DD form to DMS form
% Format of call: DMS(DD angle)
% Returns degrees, minutes, seconds

degree = floor(angles);
minutes = floor((angles - degree)*60);
seconds = ((angles - degree)*60 - minutes)*60;
end
```

6) Given the following function header:

```
function doit(a, b)
```

Which of the following function calls would be valid – and why?

```
fprintf('The result is %.1f\n', doit(4,11))
```

INVALID – Nothing is returned, so there is nothing to print

```
doit(5, 2, 11.11)
```

INVALID – too many input arguments

```
x = 11;
y = 3.3;
doit(x,y)
```

VALID

7) Write a function that prints the area and circumference of a circle for a given radius. Only the radius is passed to the function. The function does not return any values. The area is given by πr^2 and the circumference is $2 \pi r$.

printAreaCirc.m

```
function printAreaCirc(rad)
% Prints the radius, area, and circumference
%   of a circle
% Format of call: printAreaCirc(radius)
% Does not return any values

fprintf('For a circle with a radius of %.1f,\n', rad)
fprintf(' the area is %.1f and the circumference is %.1f\n',...
    pi*rad*rad, 2*pi*rad)
end
```

8) Write a function that will receive an integer *n* and a character as input arguments, and will print the character *n* times.

printNChars.m

```
function printNChars(n, ch)
% Receives n and a character and prints
% the character n times
% Format of call: printNChars(n, character)
% Does not return any values

for i = 1:n
    fprintf('%c', ch)
end
fprintf('\n')
end
```

9) Write a function that receives a matrix as an input argument, and prints a random row from the matrix.

printRanRow.m

```
function printRanRow(mat)
% Prints a random row from a matrix
% Assumes a fairly small matrix of ints
% Format of call: printRanRow(matrix)
% Does not return any values

[r c] = size(mat);

ranrow = randi([1,r]);

fprintf('%d ', mat(ranrow,:))
fprintf('\n')
end
```

10) Write a function that receives a count as an input argument, and prints the value of the count in a sentence that would read “It happened 1 time.” if the value of the count is 1, or “It happened xx times.” if the value of count (xx) is greater than 1.

printCount.m

```
function printCount(count)
% Prints a count in a correct sentence
% with an 's' for plural or not
% Format of call: printCount(count)
% Does not return any value

fprintf('It happened %d time', count)
if count > 1
    fprintf('s')
end
fprintf('.\n')
end
```

11) Write a function that receives an *x* vector, a minimum value, and a maximum value, and plots **sin(x)** from the specified minimum to the specified maximum.

plotXMinMax.m

```
function plotXMinMax(x, xmin, xmax)
% Plots sin(x) from the minimum value of x
% specified to the maximum
% Format of call: plotXMinMax(x, x minimum, x maximum)
% Does not return any values

x = linspace(xmin,xmax);
```

```
plot(x,sin(x),'*')
xlabel('x')
ylabel('sin(x)')
end
```

12) Write a function that prompts the user for a value of an integer *n*, and returns the value of *n*. No input arguments are passed to this function. Error-check to make sure that an integer is entered.

promptForN.m

```
function outn = promptForN
% This function prompts the user for n
% It error-checks to make sure n is an integer
% Format of call: promptForN or promptForN()
% Returns an integer entered by the user

inputnum = input('Enter an integer for n: ');
num2 = int32(inputnum);
while num2 ~= inputnum
    inputnum = input('Invalid! Enter an integer: ');
    num2 = int32(inputnum);
end
outn = inputnum;
end
```

13) Write a script that will:

- Call a function to prompt the user for an angle in degrees
- Call a function to calculate and return the angle in radians . (Note: π radians = 180°)
- Call a function to print the result

Write all of the functions, also. Note that the solution to this problem involves four M-files: one which acts as a main program (the script), and three for the functions.

Ch6Ex13.m

```
% Script calls functions to:
% prompt for an angle in degrees
% convert to radians
% print both

deg = promptAng;
rad = degRad(deg);
prtDegRad(deg,rad)
```

promptAng.m

```
function deg = promptAng
% Prompts for an angle in degrees
% Format of call: promptAng or promptAng()
% Returns an angle in degrees

deg = input('Enter an angle in degrees: ');
end
```

degRad.m

```
function rad = degRad(deg)
% Converts an angle from degrees to radians
% Format of call degRad(degrees)
% Returns the angle in radians

rad = deg * pi / 180;
end
```

prtDegRad.m

```
function prtDegRad(deg, rad)
% Prints an angle in degrees and radians
% Format of call: prtDegRad(degrees, radians)
% Does not return any values

fprintf('The angle %.1f degrees is \n', deg)
fprintf('equivalent to %.1f radians\n', rad)
end
```

14) Modify the program in Exercise 13 so that the function to calculate the angle is a subfunction to the function that prints.

Ch6Ex14.m

```
% Script calls functions to:
% prompt for an angle in degrees
% print degrees and radians, calling a
%    subfunction to convert to radians

deg = promptAng;
prtDegRadii(deg)
```

promptAng.m

```
function deg = promptAng
% Prompts for an angle in degrees
% Format of call: promptAng or promptAng()
% Returns an angle in degrees

deg = input('Enter an angle in degrees: ');
```

```
end
```

prtDegRadii.m

```
function prtDegRadii(deg)
% Prints an angle in degrees and radians
% Calls a subfunction to convert to radians
% Format of call: prtDegRadii(degrees)
% Does not return any values

rad = degRadii(deg);
fprintf('The angle %.1f degrees is \n', deg)
fprintf('equivalent to %.1f radians\n', rad)
end

function rad = degRadii(deg)
% Converts an angle from degrees to radians
% Format of call degRadii(degrees)
% Returns the angle in radians

rad = deg * pi / 180;
end
```

15) In 3D space, the Cartesian coordinates (x,y,z) can be converted to spherical coordinates (radius r, inclination θ, azimuth ϕ) by the following equations:

$$r = \sqrt{x^2 + y^2 + z^2}, \qquad \theta = \cos^{-1}(\frac{z}{r}), \qquad \phi = \tan^{-1}(\frac{y}{x})$$

A program is being written to read in Cartesian coordinates, convert to spherical, and print the results. So far, a script *pracscript* has been written that calls a function *getcartesian* to read in the Cartesian coordinates, and a function *printspherical* that prints the spherical coordinates. Assume that the *getcartesian* function exists and reads the Cartesian coordinates from a file. The function *printspherical* calls a subfunction *convert2spher* that converts from Cartesian to spherical coordinates. You are to write the *printspherical* function. Here is an example:

```
>> pracscript
The radius is 5.46
The inclination angle is 1.16
The azimuth angle is 1.07
```

pracscript.m

```
[x,y,z] = getcartesian();
printspherical(x,y,z)
```

getcartesian.m

```
function [x,y,z] = getcartesian()
% Assume this gets x,y,z from a file
% Function stub:
x = 11;
y = 3;
z = 7;
end
```

printspherical.m

```
function printspherical(x,y,z)
[r, inc, azi] = convert2spher(x,y,z);
 fprintf('The radius is %.2f\n', r)
fprintf('The inclination angle is %.2f\n', inc)
fprintf('The azimuth angle is %.2f\n', azi)
end

function [r, i, a] = convert2spher(x,y,z)
r = sqrt(x^2 + y^2 + z^2);
i = acos(z/r);
a = atan(y/z);
end
```

16) The lump sum S to be paid when interest on a loan is compounded annually is given by $S = P(1 + i)^n$ where *P* is the principal invested, i is the interest rate, and *n* is the number of years. Write a program that will plot the amount S as it increases through the years from 1 to *n*. The main script will call a function to prompt the user for the number of years (and error-check to make sure that the user enters a positive integer). The script will then call a function that will plot S for years 1 through n. It will use .05 for the interest rate and $10,000 for P.

Ch6Ex16.m

```
% Plots the amount of money in an account
% after n years at interest rate i with a
% principal p invested

% Call a function to prompt for n
n = promptYear;

% Call a function to plot
plotS(n, .05, 10000)
```

promptYear.m

```
function outn = promptYear
% This function prompts the user for # of years n
```

```
% It error-checks to make sure n is a positive integer
% Format of call: promptYear or promptYear()
% Returns the integer # of years

inputnum = input('Enter a positive integer for n: ');
num2 = int32(inputnum);
while num2 ~= inputnum || num2 < 0
    inputnum = input('Invalid! Enter a positive integer: ');
    num2 = int32(inputnum);
end
outn = inputnum;
end
```

plotS.m

```
function plotS(n, i, p)
% Plots the lump sum S for years 1:n
% Format of call: plotS(n,i,p)
% Does not return any values

vec = 1:n;
s = p * (1+i).^ vec;
plot(vec,s,'k*')
xlabel('n (years)')
ylabel('S')
end
```

17) Write a program to write a length conversion chart to a file. It will print lengths in feet, from 1 to an integer specified by the user, in one column and the corresponding length in meters (1 foot = 0.3048 m) in a second column. The main script will call one function that prompts the user for the maximum length in feet; this function must error-check to make sure that the user enters a valid positive integer. The script then calls a function to write the lengths to a file.

Ch6Ex17.m

```
% Write a length conversion chart to a file

% Call a function to get the max length in feet
maxl = promptMaxL;

% Call a function to write the chart to a file
lengthChart(maxl)
```

promptMaxL.m

```
function maxl = promptMaxL
% This function prompts the user for a max length in feet
```

```
% It error-checks to make sure it is a positive integer
% Format of call: promptmaxl or promptmaxl()
% Returns the integer maximum length in feet

inputnum = input('Enter the maximum length: ');
num2 = int32(inputnum);
while num2 ~= inputnum | num2 < 0
    inputnum = input('Invalid! Enter a positive integer: ');
    num2 = int32(inputnum);
end
maxl = inputnum;
end
```

lengthChart.m

```
function lengthChart(maxl)
% Creates a chart of converting feet to
% meters and writes it to a file
% Format of call: lengthChart(maximum length)
% Does not return any values

ft = 1:maxl;
meters = ft * .3048;

chart = [ft;meters]';

save ftmetchart.dat chart -ascii
end
```

18) The script *circscript* loops n times to prompt the user for the circumference of a circle (where n is a random integer). Error-checking is ignored to focus on functions in this program. For each, it calls one function to calculate the radius and area of that circle, and then calls another function to print these values. The formulas are $r = c/(2\Pi)$ and $a = \Pi r^2$ where r is the radius, c is the circumference, and a is the area. Write the two functions.

circscript.m

```
n = randi(4);
for i = 1:n
   circ = input('Enter the circumference of the circle: ');
   [rad, area] = radarea(circ);
   dispra(rad,area)
end
```

radarea.m

```
function [radius, area] = radarea(circ)
```

```
% Calculates the radius and area of a circle,
% given the circumference
% Format of call: radarea(circumference)
% Returns the radius then area

radius = circ/(2*pi);
area = pi * radius ^2;
end
```

dispra.m

```
function dispra(radius, area)
% Prints the radius and area of a circle
% Format of call: dispra(radius, area)
% Does not return any values

fprintf('The radius of the circle is %.2f\n', radius)
fprintf('The area of the circle is %.2f\n', area)
end
```

19) The distance between any two points (x_1,y_1) and (x_2,y_2) is given by:

distance = $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

The area of a triangle is:

area = $\sqrt{s*(s-a)*(s-b)*(s-c)}$

where a, b, and c are the lengths of the sides of the triangle, and s is equal to half the sum of the lengths of the three sides of the triangle. Write a script that will prompt the user to enter the coordinates of three points that determine a triangle (e.g. the x and y coordinates of each point). The script will then calculate and print the area of the triangle. It will call one function to calculate the area of the triangle. This function will call a subfunction that calculates the length of the side formed by any two points (the distance between them).

Ch6Ex19.m

```
% Calculate the area of a triangle given the
% coordinates of the 3 points that determine it

% Prompt the user for the coordinates of the points

x1 = input('Enter the x coordinate of point 1: ');
y1 = input('Enter the y coordinate of point 1: ');
x2 = input('Enter the x coordinate of point 2: ');
y2 = input('Enter the y coordinate of point 2: ');
x3 = input('Enter the x coordinate of point 3: ');
y3 = input('Enter the y coordinate of point 3: ');
```

```
% Call a function to calculate the area, then print it
area = triarea(x1,y1,x2,y2,x3,y3);
fprintf('The area of the triangle is %.2f\n', area)
```

triarea.m

```
function outarea = triarea(x1,y1,x2,y2,x3,y3)
% Calculates the area of the triangle
% Format of call: triarea(x1,y1,x2,y2,x3,y3)
% Returns the area of the triangle

a = dist(x1,y1,x2,y2);
b = dist(x2,y2,x3,y3);
c = dist(x3,y3,x1,y1);
s = 0.5*(a+b+c);

outarea = sqrt(s*(s-a)*(s-b)*(s-c));
end

function outd = dist(x1,y1,x2,y2)
% Calculates the distance between any two points
% Format of call: dist(x1,y1,x2,y2)
% Returns the distance between the two points

outd = sqrt((x1-x2)^2 + (y1-y2)^2);
end
```

20) Write a program to write a temperature conversion chart to a file. The main script will:

- call a function that explains what the program will do
- call a function to prompt the user for the minimum and maximum temperatures in degrees Fahrenheit, and return both values. This function checks to make sure that the minimum is less than the maximum, and calls a subfunction to swap the values if not.
- call a function to write temperatures to a file: the temperature in degrees F from the minimum to the maximum in one column, and the corresponding temperature in degrees Celsius in another column. The conversion is C = (F – 32) * 5/9.

Ch6Ex20.m

```
% Writes a temperature conversion chart to a file

% Explain the program
explainTemp

% Prompt the user for min and max F temps
```

```
[tmin, tmax] = tempMinMax;

% Write the F temps and corresponding C temps to file
fandCTemps(tmin, tmax)
```

explainTemp.m

```
function explainTemp
% Explains that the program will write a temperature
% conversion chart from F to C to a file

fprintf('This program writes a temperature conversion')
fprintf(' chart to a file fcchart.dat.\n')
fprintf('\nIt prompts the user for a minimum and maximum')
fprintf(' temp in degrees F\n')
fprintf('It writes the F temps from min to max in one')
fprintf(' column\n and correspoding C temps in another\n')
end
```

tempMinMax.m

```
function [tmin, tmax] = tempMinMax
% Prompts the user for min and max F temps
% and makes sure min < max
% Format of call: tempMinMax or tempMinMax()
% Returns min and max temperatures in F

tmin = input('Enter the minimum F temp: ');
tmax = input('Enter the maximum F temp: ');

if tmin > tmax
   [tmin, tmax] = swap(tmin, tmax);
end
end

function [outa, outb] = swap(ina, inb)
% swaps the values of arguments
% Format of call: swap(a,b)
% Returns b then a

outa = inb;
outb = ina;
end
```

fandCTemps.m

```
function fandCTemps(tmin, tmax)
% Writes the F and C temps to a file
% Format of call: fandCTemps(min temp, max temp)
```

```
% Does not return any values

f = tmin:tmax;
c = (f-32)*5/9;

mat = [f;c]';

save fcchart.dat mat -ascii
end
```

21) Modify the function *func2* from Section 6.4.1 that has a **persistent** variable *count*. Instead of having the function print the value of *count*, the value should be returned.

func2ii.m

```
function outc = func2ii
% Returns the value of a persistent variable
% that counts the # of times the function is called
% Format of call: func2ii or func2ii()
% Returns the value of the persistent count

persistent count

if isempty(count)
    count = 0;
end
count = count + 1;
outc = count;
end
```

22) Write a function *per2* that receives one number as an input argument. The function has a **persistent** variable that sums the values passed to it. Here are the first two times the function is called:

```
>> per2(4)
ans =
     4

>> per2(6)
ans =
    10
```

per2.m

```
function outstat = per2(num)
% Persistent variable sums the numbers
% passed to this function
% Format of call: per2(input number)
```

```
% Returns the persistent sum of input arguments

persistent mysum

if isempty(mysum)
    mysum = 0;
end
mysum = mysum + num;
outstat = mysum;
end
```

23) What would be the output from the following program? Think about it, write down your answer, and then type it in to verify.

testscope.m

```
answer = 5;
fprintf('Answer is %d\n',answer)
pracfn
pracfn
fprintf('Answer is %d\n',answer)
printstuff
fprintf('Answer is %d\n',answer)
```

pracfn.m

```
function pracfn
persistent count
if isempty(count)
    count = 0;
end
count = count + 1;
fprintf('This function has been called %d times.\n',count)
end
```

printstuff.m

```
function printstuff
answer = 33;
fprintf('Answer is %d\n',answer)
pracfn
fprintf('Answer is %d\n',answer)
end
```

```
>> testscope
Answer is 5
This function has been called 1 times.
This function has been called 2 times.
Answer is 5
```

```
Answer is 33
This function has been called 3 times.
Answer is 33
Answer is 5
>>
```

24) Assume a matrix variable *mat*, as in the following example:

```
mat =
     4     2     4     3     2
     1     3     1     0     5
     2     4     4     0     2
```

The following **for** loop

```
[r, c] = size(mat);
for i = 1:r
    sumprint(mat(i,:))
end
```

prints this result:

```
The sum is now 15
The sum is now 25
The sum is now 37
```

Write the function *sumprint*.

sumprint.m

```
function sumprint(vec)
% Prints the result of a persistent sum
% of the values in vectors that are passed
% Format of call: sumprint(vector)
% Returns the persistent sum of all input vectors

persistent mysum

if isempty(mysum)
    mysum = 0;
end

mysum = mysum + sum(vec);
fprintf('The sum is now %d\n', mysum)
end
```

25) The following script *land* calls functions to:

- prompt the user for a land area in acres
- calculate and return the area in hectares and in square miles
- print the results

One acre is 0.4047 hectares. One square mile is 640 acres. Assume that the last function, that prints, exists – you do not have to do anything for that function. You are to write the entire function that calculates and returns the area in hectares and in square miles, and write just a function stub for the function that prints and reads. Do NOT write the actual contents of this function; just write a stub!

land.m

```
inacres = askacres;
[sqmil, hectares] = convacres(inacres);
dispareas(inacres, sqmil, hectares) % Assume this exists
```

askacres.m

```
function acres = askacres
acres = 33;
end
```

convacres.m

```
function [sqmil, hectares] = convacres(inacres)
sqmil = inacres/640;
hectares = inacres*.4047;
end
```

26) The braking distance of a car depends on its speed as the brakes are applied and on the car's braking efficiency. A formula for the braking distance is

$$b_d = \frac{s^2}{2Rg}$$

where b_d is the braking distance, s is the car's speed, R is the braking efficiency and g is the acceleration due to gravity (9.81). A script has been written that calls a function to prompt the user for s and R, calls another function to calculate the braking distance, and calls a third function to print the braking distance in a sentence format with one decimal place. You are to write a function stub for the function that prompts for s and R, and the actual function definitions for the other two functions.

Ch6Ex26.m

```
[s, R] = promptSandR;
brakDist = calcbd(s, R);
printbd(brakDist)
```

promptSandR.m

```
function [s, R] = promptSandR
s = 33;
```

```
R = 11;
end
```

calcbd.m

```
function bd = calcbd(s, R)
bd = s .^ 2 / (2*R*9.81);
end
```

printbd.m

```
function printbd(bd)
fprintf('The braking distance was %.1f\n', bd)
end
```

27) Write a menu-driven program to convert a time in seconds to other units (minutes, hours, and so on). The main script will loop to continue until the user chooses to exit. Each time in the loop, the script will generate a random time in seconds, call a function to present a menu of options, and print the converted time. The conversions must be made by individual functions (e.g. one to convert from seconds to minutes). All user-entries must be error-checked.

Ch6Ex27.m

```
% Menu-driven program to convert a time from seconds
% to other units specified by the user

% Generate a random time in seconds
rantime = randi([1 10000]);
% Call a function to display a menu and get a choice
choice = timeoption;

% Choice 3 is to exit the program
while choice ~= 3
   switch choice
       case 1
           fprintf('%d seconds is %.1f minutes\n',...
               rantime, secsToMins(rantime));
       case 2
           fprintf('%d seconds is %.2f hours\n',...
               rantime, secsToHours(rantime));
   end
   % Generate a random time in seconds
   rantime = randi([1 10000]);
   % Display menu again and get user's choice
   choice = timeoption;
end
```

timeoption.m

```
function choice = timeoption
% Print the menu of options and error-check
%  until the user pushes one of the buttons
% Format of call: timeoption or timeoption()
% Returns the integer value of the choice, 1-3

choice = menu('Choose a unit', 'Minutes', ...
    'Hours', 'Exit');
% If the user closes the menu box rather than
%  pushing one of the buttons, choice will be 0
while choice == 0
    disp('Error - please choose one of the options.')
    choice = menu('Choose a unit', 'Minutes', ...
    'Hours', 'Exit');
end
end
```

secsToMins.m

```
function mins = secsToMins(seconds)
% Converts a time from seconds to minutes
% Format secsToMins(seconds)
% Returns the time in minutes

mins = seconds / 60;
end
```

secsToHours.m

```
function hours = secsToHours(seconds)
% Converts a time from seconds to hours
% Format secsToHours(seconds)
% Returns the time in hours

hours = seconds / 3600;
end
```

28) Write a menu-driven program to investigate the constant π. Model it after the program that explores the constant *e*. Pi (π) is the ratio of a circle's circumference to its diameter. Many mathematicians have found ways to approximate π. For example, Machin's formula is:

$$\frac{\pi}{4} = 4 \arctan\left(\frac{1}{5}\right) - \arctan\left(\frac{1}{239}\right)$$

Leibniz found that π can be approximated by:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \ldots$$

This is called a sum of a series. There are six terms shown in this series. The first term is 4, the second term is –4/3, the third term is 4/5, and so forth. For example, the menu-driven program might have the following options:

- Print the result from Machin's formula.
- Print the approximation using Leibniz' formula, allowing the user to specify how many terms to use.
- Print the approximation using Leibniz' formula, looping until a "good" approximation is found.
- Exit the program.

Ch6Ex28.m

```
% This script explores pi

% Call a function to display a menu and get a choice
choice = pioption;

% Choice 4 is to exit the program
while choice ~= 4
   switch choice
       case 1
           % Print result from Machin's formula
           pimachin
       case 2
           % Approximate pi using Leibniz,
           % allowing user to specify # of terms
           pileibnizn
       case 3
           % Approximate pi using Leibniz,
           %  until a "good" approximation is found
           pileibnizgood
   end
   % Display menu again and get user's choice
   choice = pioption;
end
```

pioption.m

```
function choice = pioption
% Print the menu of options and error-check
%  until the user pushes one of the buttons
% Format of call: pioption or pioption()
% Returns integer of user's choice, 1-4

```

```
choice = menu('Choose a pi option', 'Machin', ...
    'Leibniz w/ n', 'Leibniz good', 'Exit Program');
% If the user closes the menu box rather than
%  pushing one of the buttons, choice will be 0
while choice == 0
    disp('Error - please choose one of the options.')
    choice = menu('Choose a pi option', 'Machin', ...
    'Leibniz w/ n', 'Leibniz good', 'Exit Program');

end
end
```

pimachin.m

```
function pimachin
% Approximates pi using Machin's formula and prints it
% Format of call: pimachin or pimachin()
% Does not return any values

machinform = 4 * atan(1/5) - atan(1/239);

fprintf('Using the MATLAB constant, pi = %.6f\n', pi)
fprintf('Using Machin''s formula, pi = %.6f\n',4*machinform)
end
```

pileibnizn.m

```
function pileibnizn
% Approximates and prints pi using Leibniz' formula
% Prompt user for number of terms n
% Format of call: pileibnizn or pileibnizn()
% Does not return any values

fprintf('Approximate pi using Leibiz'' formula\n')

% Call a subfunction to prompt user for n
n = askforn;

approxpi = 0;
denom = -1;
termsign = -1;
for i = 1:n
    denom = denom + 2;
    termsign = -termsign;
    approxpi = approxpi + termsign * (4/denom);
end
fprintf('An approximation of pi with n = %d is %.2f\n', ...
    n, approxpi)
```

```
end

function outn = askforn
% This function prompts the user for n
% It error-checks to make sure n is a positive integer
% Format of call: askforn or askforn()
% Returns positive integer n

inputnum = input('Enter a positive integer for n: ');
num2 = int32(inputnum);
while num2 ~= inputnum | num2 < 0
   inputnum = input('Invalid! Enter a positive integer: ');
   num2 = int32(inputnum);
end
outn = inputnum;
end
```

29) Write a program to calculate the position of a projectile at a given time t. For an initial velocity v_0 and angle of departure θ_0, the position is given by x and y coordinates as follows (note: the gravity constant g is 9.81m/s^2):

$$x = v_0 \cos(\theta_0)t$$

$$y = v_0 \sin(\theta_0)t - \frac{1}{2}gt^2$$

The program should initialize the variables for the initial velocity, time, and angle of departure. It should then call a function to find the x and y coordinates, and then another function to print the results. If you have version R2016a or later, make the script into a live script.

Ch6Ex29.m

```
% Projectile motion

v0 = 33;
theta0 = pi/4;

[x, y] = findcoords(v0, theta0, t);

printcoords(x,y)
```

findcoords.m

```
function [x, y] = findcoords(v0, theta0, t)
```

```
g = 9.81;

x = v0 * cos(theta0) * t;
y = v0 * sin(theta0) * t - 0.5 * g * t * t;
end
```

printcoords.m

```
function printcoords(x,y)
fprintf('x is %f and y is %f\n', x, y)
end
```

Chapter 7: String Manipulation

Exercises

1) A file name is supposed to be in the form *filename.ext*. Write a function that will determine whether a string is in the form of a name followed by a dot followed by a three-character extension, or not. The function should return 1 for **logical true** if it is in that form, or 0 for **false** if not.

fileNameCheck.m

```
function out = fileNameCheck(str)
% Checks to see if input is in the proper filename format
% Format of call: fileNameCheck(string)
% Returns true if in form filename.ext, false if not

out = str(end-3) == '.';

end
```

2) The following script calls a function *getstr* that prompts the user for a string, error-checking until the user enters something (the error would occur if the user just hits the Enter key without any other characters first). The script then prints the length of the string. Write the *getstr* function.

Ch7Ex2.m

```
thestring = getstr();
fprintf('Thank you, your string is %d characters long\n', ...
    length(thestring))
```

getstr.m

```
function outstr = getstr
```

```
% Prompts the user until the user enters a string
% with a length > 0
% Format of call: getstring or getstring()

outstr = input('Please enter a string: ', 's');
while isempty(outstr)
    outstr = input('PLEASE enter a string: ', 's');
end
end
```

3) Write a script that will, in a loop, prompt the user for four course numbers. Each will be a string of length 5 of the form 'CS101'. These strings are to be stored in a character matrix.

Ch7Ex3.m

```
% Creates a character matrix with four course numbers

courses = [];
for i = 1:4
    courses = char(courses, ...
            input('Enter a course number: ','s'));
end
```

4) Write a function that will generate two random integers, each in the inclusive range from 10 to 30. It will then return a string consisting of the two integers joined together, e.g. if the random integers are 11 and 29, the string that is returned will be '1129'.

twoRanAsStr.m

```
function outstr = twoRanAsStr
% Creates a string consisting of two random
% integers, concatenated together
% Format of call: twoRanAsStr or twoRanAsStr()
% Returns a 4-character string

ran1 = randi([10,30]);
ran2 = randi([10,30]);

outstr = strcat(int2str(ran1),int2str(ran2));
end
```

5) Write a script that will create x and y vectors. Then, it will ask the user for a color ('red', 'blue', or 'green') and for a plot style (circle or star). It will then create a string *pstr* that contains the color and plot style, so that the call to the **plot** function would be: **plot(x,y,pstr)**.

For example, if the user enters ‘blue’ and ‘*’, the variable *pstr* would contain ‘b*’.

Ch7Ex5.m

```
% Create x and y vectors.  Prompt the user for a color and
% plot style for plotting the x and y vectors

x = -5: 0.5: 5;
y = sin(x);
color = input('Enter a color; red, green, or blue: ','s');
style = input('Enter a plot style; * or o: ','s');
pstr = strcat(color(1),style);
plot(x,y,pstr)
```

6) Assume that you have the following function and that it has not yet been called.

strfunc.m

```
function strfunc(instr)
persistent mystr
if isempty(mystr)
    mystr = '';
end
mystr = strcat(instr,mystr);
fprintf('The string is %s\n',mystr)
end
```

What would be the result of the following sequential expressions?

```
strfunc('hi')

strfunc('hello')
```

```
>> strfunc('hi')
The string is hi
>> strfunc('hello')
The string is hellohi
>>
```

7) Explain in words what the following function accomplishes (not step-by-step, but what the end result is).

dostr.m

```
function out = dostr(inp)
persistent str
[w, r] = strtok(inp);
```

```
    str = strcat(str,w);
out = str;
end
```

It takes the first word of every sentence passed to it, and concatenates it to a string.

8) Write a function that will receive a name and department as separate strings and will create and return a code consisting of the first two letters of the name and the last two letters of the department. The code should be upper-case letters. For example,

```
>> namedept('Robert','Mechanical')
ans =
ROAL
```

nameDept.m

```
function outcode = nameDept(name, department)
% Creates a code from a name and department
% consisting of first two letters of the name
% and the last two of the department, upper case
% Format of call: nameDept(name, department)
% Returns the 4-character code

outcode = name(1:2);
outcode = upper(strcat(outcode, department(end-1:end)));
end
```

9) Write a function “createUniqueName” that will create a series of unique names. When the function is called, a string is passed as an input argument. The function adds an integer to the end of the string, and returns the resulting string. Every time the function is called, the integer that it adds is incremented. Here are some examples of calling the function:

```
>> createUniqueName('hello')
ans =
hello1
>> varname = createUniqueName('variable')
varname =
variable2
```

createUniqueName.m

```
function newfilename = createUniqueName(instr)
persistent count
if isempty(count)
    count = 0;
```

```
end
count = count + 1;
newfilename = strcat(instr,int2str(count));
end
```

10) What does the **blanks** function return when a 0 is passed to it? A negative number? Write a function *myblanks* that does exactly the same thing as the **blanks** function, using the programming method. Here are some examples of calling it:

```
     >> fprintf('Here is the result:%s!\n', myblanks(0))
     Here is the result:!

     >> fprintf('Here is the result:%s!\n', myblanks(7))
     Here is the result:       !
```

```
>> nob = blanks(0)
nob =
   Empty string: 1-by-0
>> nob = blanks(-4)
nob =
   Empty string: 1-by-0
>>
```

myblanks.m

```
function outa = myblanks(num)
% Mimics the blanks function
% Format of call: myblanks(n)
% Returns a string of n blanks

outa = '';
if num > 0
    for i = 1:num
        outa = [outa ' '];
    end
end
end
```

11) Write a function that will prompt the user separately for a filename and extension and will create and return a string with the form 'filename.ext'.

getfilename.m

```
function file = getfilename
% Prompts user for filename and extension and combines them
% Format of call: getfilename or getfilename()
```

```
% Returns one string of the form filename.ext

fname = input('Enter filename: ','s');
extension = input('Enter extension: ','s');
file = sprintf('%s.%s',fname,extension);
end
```

12) Write a function that will receive one input argument, which is an integer *n*. The function will prompt the user for a number in the range from 1 to *n* (the actual value of *n* should be printed in the prompt) and return the user's input. The function should error-check to make sure that the user's input is in the correct range.

pickInRange.m

```
function choice = pickInRange(n)
%Prompts user to enter an integer in the range
% from 1 to the input argument n
%Error-checks to make sure input is in range
% Format of call: pickInRange(n)
% Returns one integer in the range 1-n

prompt = ...
    sprintf('Enter an integer in the range from 1 to %d: ',n);

choice = input(prompt);

while choice < 1 || choice > n
    disp('Error! Value not in range.')
    choice = input(prompt);
end
end
```

13) Write a script that will generate a random integer, ask the user for a field width, and print the random integer with the specified field width. The script will use **sprintf** to create a string such as 'The # is %4d\n' (if, for example, the user entered 4 for the field width) which is then passed to the **fprintf** function. To print (or create a string using **sprintf**) either the % or \ character, there must be two of them in a row.

Ch7Ex13.m

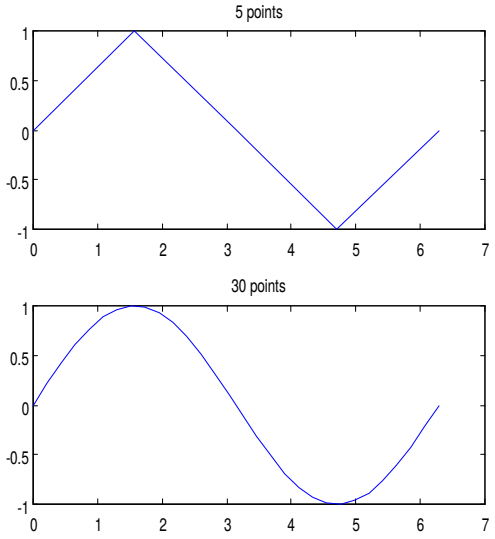
```
% Generate a random integer, prompt the user for
% a field width, and print the integer in that width

integer = randi([0 100]);
width = input('Enter the field width: ');
```

```
fprintf(sprintf('The # is %%%dd\n',width),integer)
```

14) Write an m-file function called “plotsin” that will graphically demonstrate the difference in plotting the sin function with a different number of points in the range from 0 to 2 Π. The function will receive two arguments, which are the number of points to use in two different plots of the sin function. For example, the following call to the function:

```
>> plotsin(5,30)
```

will result in the following figure window in which the first plot has 5 points altogether in the range from 0 to 2 Π, inclusive, and the second has 30:



plotsin.m

```
function plotsin(lowp, highp)
x = linspace(0,2*pi,lowp);
y = sin(x);
subplot(2,1,1)
plot(x,y)
title(sprintf('%d points',lowp))
subplot(2,1,2)
x = linspace(0,2*pi,highp);
y = sin(x);
plot(x,y)
title(sprintf('%d points',highp))
end
```

15) If the strings passed to **strfind** are the same length, what are the only two possible results that could be returned?

```
1 or []
```

16) Vectorize this:

```
while mystrn(end) == ' '   % Note one space in quotes
     mystrn = mystrn(1:end-1);
end
```

```
mystrn = deblank(mystrn);
```

17) Vectorize this:

```
loc = findstr(sentence, ' ');

where = loc(1);

first = sentence(1:where-1);

last = sentence(where:end);
```

```
[first, last] = strtok(sentence);
```

18) Vectorize this:

```
vec = [];
for i = 1:8
   vec = [ vec ' ];  % one blank space
end
vec   % just for display
```

```
vec = blanks(8)
```

19) Vectorize this:

```
if length(str1) ~= length(str2)
    outlog = false;
else
```

```
    outlog = true;
    for i=1:length(str1)
        if str1(i) ~= str2(i)
            outlog = false;
        end
    end
end
outlog % Just to display the value
```

```
     outlog = strcmp(str1, str2)

OR:

     outlog = all(str1 == str2)
```

20) Write a function *nchars* that will create a string of n characters, without using any loops or selection statements.

```
     >> nchars('*', 6)
     ans =
     ******
```

nchars.m

```
function outstr = nchars(onechar, n)

outstr = blanks(n);
outstr = strrep(outstr,' ',onechar);
end
```

21) Write a function that will receive two input arguments: a character matrix that is a column vector of strings, and a string. It will loop to look for the string within the character matrix. The function will return the row number in which the string is found if it is in the character matrix, or the empty vector if not. Use the programming method.

loopstring.m

```
function index = loopstring(charmat,str)
% Loops through input character matrix searching for string
% Format of call: loopstring(character matrix, string)
% Returns indices of string in charmat, or empty vector if not
found

[r c] = size(charmat);
index = [];
```

```
for i = 1:r
    if strcmp(strtrim(charmat(i,:)),str)
        index = [index i];
    end
end
end
```

22) Write a function *rid_multiple_blanks* that will receive a string as an input argument. The string contains a sentence that has multiple blank spaces in between some of the words. The function will return the string with only one blank in between words. For example,

```
>> mystr = 'Hello   and how   are   you?';
>> rid_multiple_blanks(mystr)
ans =
Hello and how are you?
```

rid_multiple_blanks.m

```
function newstr = rid_multiple_blanks(str)
% Deletes multiple blanks from a string
% Format of call: rid_multiple_blanks(input string)
% Returns a string with multiple blanks replaced
% by a single blank space

while length(strfind(str,'  ')) > 0
    str = strrep(str,'  ', ' ');
end
newstr = str;
end
```

23) Words in a string variable are separated by right slashes (/) instead of blank spaces. Write a function *slashtoblank* that will receive a string in this form and will return a string in which the words are separated by blank spaces. This should be general and work regardless of the value of the argument. No loops allowed in this function; the built-in string function(s) must be used.

slashToBlank.m

```
function outstr = slashToBlank(instr)
% Replaces slashes with blanks
% Format of call: slashToBlank(input string)
% Returns new string w/ blanks replacing /

outstr = strrep(instr,'/',' ');
end
```

24) Two variables store strings that consist of a letter of the alphabet, a blank space, and a number (in the form ‘R 14.3’). Write a script that would initialize two such variables. Then, use string manipulating functions to extract the numbers from the strings and add them together.

Ch7Ex24.m

```
% Creates two strings in the form 'R 14.3', extracts
%  the numbers, and adds them together

str1 = 'k 8.23';
str2 = 'r 24.4';

[letter1, rest] = strtok(str1);
num1 = str2num(rest);

[letter2, rest] = strtok(str2);
num2 = str2num(rest);

num1+num2
```

Cryptography, or encryption, is the process of converting plaintext (e.g., a sentence or paragraph), into something that should be unintelligible, called the ciphertext. The reverse process is code-breaking, or cryptanalysis, which relies on searching the encrypted message for weaknesses and deciphering it from that point. Modern security systems are heavily reliant on these processes.

25) In cryptography, the intended message sometimes consists of the first letter of every word in a string. Write a function *crypt* that will receive a string with the encrypted message and return the message.

```
>> estring = 'The early songbird tweets';
>> m = crypt(estring)
m =
Test
```

crypt.m

```
function message = crypt(instring);
% This function returns the message hidden in the
%  input string, which is the first letter of
%  every word in the string
% Format crypt(input string)
% Returns a string consisting of the first letter
% of every word in the input string
```

```
rest = strtrim(instring);
message = '';
while ~isempty(rest)
   [word, rest] = strtok(rest);
   message = strcat(message,word(1));
end
end
```

26) Using the functions **char** and **double**, one can shift words. For example, one can convert from lower case to upper case by subtracting 32 from the character codes:

```
>> orig = 'ape';
>> new = char(double(orig)-32)
new =
APE

>> char(double(new)+32)
ans =
ape
```

We've "encrypted" a string by altering the character codes. Figure out the original string. Try adding and subtracting different values (do this in a loop) until you decipher it:

```
Jmkyvih$mx$syx$}ixC
```

Ch7Ex26.m

```
% Loop to decipher the code

code = 'Jmkyvih$mx$syx$}ixC';

code = char(double(code)-1);
disp(code)
choice = input('Enter ''c'' to continue: ','s');

while choice == 'c'
    code = char(double(code)-1);
    disp(code)
    choice = input('Enter ''c'' to continue: ','s');
end
```

```
>> Ch7Ex26
Iljxuhg#lw#rxw#|hwB
Enter 'c' to continue: c
Hkiwtgf"kv"qwv"{gvA
Enter 'c' to continue: c
Gjhvsfe!ju!pvu!zfu@
Enter 'c' to continue: c
```

```
Figured it out yet?
Enter 'c' to continue: yes
>>
```

27) Load files named *file1.dat*, *file2.dat*, and so on in a loop. To test this, create just two files with these names in your Current Folder first.

Ch7Ex27.m

```
for i = 1:2
    eval(sprintf('load file%d.dat',i))
end
```

28) Either in a script or in the Command Window, create a string variable that stores a string in which numbers are separated by the character 'x', for example '12x3x45x2'. Create a vector of the numbers, and then get the sum (e.g., for the example given it would be 62 but the solution should be general).

```
>> str = '12x3x45x2';
>> vec = str2num(strrep(str,'x',' '));
>> sum(vec)
ans =
    62
```

29) Create the following two variables:

```
>> var1 = 123;
>> var2 = '123';
```

Then, add 1 to each of the variables. What is the difference?

```
>> var1 + 1      % 123 + 1 --> 124
ans =
   124
>> var2 + 1      % '1'+1-->'2', '2'+1-->'3', '3'+1-->'4'
ans =
    50    51    52
>> char(ans)
ans =
234
```

30) The built-in **clock** function returns a vector with six elements representing the year, month, day, hours, minutes and seconds. The first five elements are integers whereas the last is a **double** value, but calling it with **fix** will convert all to integers. The built-in **date** function returns the day, month, and year as a string. For example,

```
>> fix(clock)
ans =
```

```
        2013       4       25       14       25       49
```

```
>> date
ans =
25-Apr-2013
```

Write a script that will call both of these built-in functions, and then compare results to make sure that the year is the same. The script will have to convert one from a string to a number, or the other from a number to a string in order to compare.

Ch7Ex30.m

```
% Compares years obtained from built-in functions
% date and clock to ensure they are the same

c = fix(clock);
d = date;

dyear = str2double(d(end-3:end));

if dyear == c(1)
    disp('Years are the same!')
else
    disp('Years are not the same.')
end
```

31) Use **help isstrprop** to find out what properties can be tested; try some of them on a string variable.

```
>> let = 'x';
>> word = 'hi123';
>> isstrprop(let,'alpha')
ans =
     1
>> isstrprop(word,'alpha')
ans =
     1     1     0     0     0
>> isstrprop(word,'alphanum')
ans =
     1     1     1     1     1
```

32) Find out how to pass a vector of integers to **int2str** or real numbers to **num2str**.

```
>> intString = int2str(2:5)
```

```
intString =
2  3  4  5
>> length(intString)
ans =
    10
>> str2num(intString(end-1:end))
ans =
     5
>> realString = num2str([11.11 33.3])
realString =
11.11         33.3
```

33) Write a script that will first initialize a string variable that will store x and y coordinates of a point in the form ‘x 3.1 y 6.4’. Then, use string manipulating functions to extract the coordinates and plot them.

Ch7Ex33.m

```
% create a string variable of the form 'x 3.1 y 6.4'
% extract the x and y coordinates and plot

str = 'x 2.3 y 4.5';
[letter, rest] = strtok(str);
[x, rest] = strtok(rest);
[letter, rest] = strtok(rest);
y = rest(2:end);

x = str2num(x);
y = str2num(y);

plot(x,y,'go')
```

34) Write a function *wordscramble* that will receive a word in a string as an input argument. It will then randomly scramble the letters and return the result. Here is an example of calling the function:

```
>> wordscramble('fantastic')
ans =
safntcait
```

wordScramble.m

```
function outword = wordScramble(inword)
% Randomly scramble the letters in a word
% Format of call: wordScramble(word)
% Returns string with same length as input
%  string but with characters randomly scrambled

len = length(inword);
```

```
% Put random index in first element
indvec = zeros(1,len);
indvec(1) = randi([1 len]);

% Make sure every index only used once
for i = 2:len
   ran = randi([1 len]);
   while any(indvec(1:i-1)== ran)
      ran = randi([1,len]);
   end
   indvec(i) = ran;
end
outword = inword(indvec);
end
```

Massive amounts of temperature data have been accumulated and stored in files. To be able to comb through this data and gain insights into global temperature variations, it is often useful to visualize the information.

35) A file called *avehighs.dat* stores for three locations the average high temperatures for each month for a year (rounded to integers). There are three lines in the file; each stores the location number followed by the 12 temperatures (this format may be assumed). For example, the file might store:

```
432  33 37 42 45 53 72 82 79 66 55 46 41
777  29 33 41 46 52 66 77 88 68 55 48 39
567  55 62 68 72 75 79 83 89 85 80 77 65
```

Write a script that will read these data in and plot the temperatures for the three locations separately in one Figure Window. A **for** loop must be used to accomplish this. For example, if the data are as shown in the previous data block, the Figure Window would appear. The axis labels and titles should be as shown.

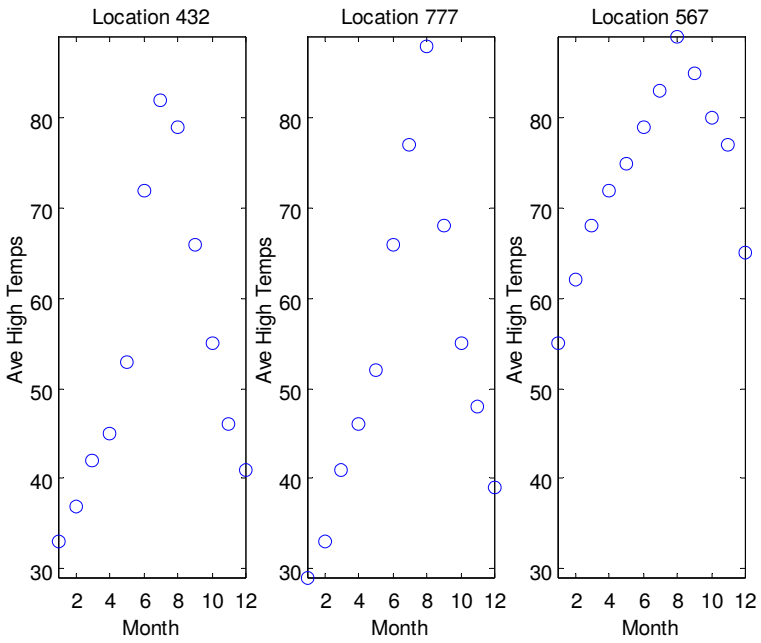


Figure Subplot to display data from file using a **for** loop

If you have version R2016a or later, write the script as a live script.

Ch7Ex35.m

```
% Use subplot to plot temperature data from
% 3 locations

load avehighs.dat
for i = 1:3
    loc = avehighs(i,1);
    temps = avehighs(i,2:13);
    subplot(1,3,i)
    plot(1:12,temps,'bo')
    title(sprintf('Location %d',loc))
    xlabel('Month')
    ylabel('Ave High Temps')
    mintemp = min(min(avehighs(1:3,2:13)));
    maxtemp = max(max(avehighs(1:3,2:13)));
    axis([1 12 mintemp maxtemp])
end
```

Chapter 8: Data Structures: Cell Arrays, Structures

Exercises

1) Create the following cell array:

```
>> ca = {'abc', 11, 3:2:9, zeros(2)}
```

Use the **reshape** function to make it a *2 x 2* matrix. Then, write an expression that would refer to just the last column of this cell array.

```
>> ca = reshape(ca,2,2);
>> ca{:,2}
```

2) Create a *2 x 2* cell array using the **cell** function and then put values in the individual elements. Then, insert a row in the middle so that the cell array is now *3 x 2*.

```
>> mycell = cell(2);
>> mycell{1} = 'EK';
>> mycell{2} = 127;
>> mycell{3} = 1:3;
>> mycell{4} = 'A1'
mycell =
    'EK'     [1x3 double]
    [127]    'A1'
>> mycell(3,:) = mycell(2,:)
mycell =
    'EK'     [1x3 double]
    [127]    'A1'
    [127]    'A1'
>> mycell{2,1} = 'hi';
>> mycell{2,2} = 'there'
mycell =
    'EK'     [1x3 double]
    'hi'     'there'
    [127]    'A1'
```

3) Create a row vector cell array to store the string ‘xyz’, the number 33.3, the vector 2:6, and the **logical** expression ‘a’ < ‘c’. Use the transpose operator to make this a column vector, and use **reshape** to make it a 2 x 2 matrix. Use **celldisp** to display all elements.

```
>> mycell = {'xyz',33.3,2:6,'a' < 'c'}
mycell =
    'xyz'    [33.3000]    [1x5 double]    [1]

>> mycell = mycell'
mycell =
    'xyz'
    [   33.3000]
    [1x5 double]
    [         1]

>> mycell = reshape(mycell,2,2)
```

```
mycell =
    'xyz'         [1x5 double]
    [33.3000]     [          1]

>> celldisp(mycell)
mycell{1,1} =
xyz

mycell{2,1} =
   33.3000

mycell{1,2} =
     2     3     4     5     6

mycell{2,2} =
     1
```

4) Create a cell array that stores phrases, such as:

```
exclaimcell = {'Bravo', 'Fantastic job'};
```

Pick a random phrase to print.

```
>> ranindex = randi([1, length(exclaimcell)]);
>> fprintf('%s\n', exclaimcell{ranindex})
```

5) Create three cell array variables that store people's names, verbs, and nouns. For example,

```
names = {'Harry', 'Xavier', 'Sue'};
verbs = {'loves', 'eats'};
nouns = {'baseballs', 'rocks', 'sushi'};
```

Write a script that will initialize these cell arrays, and then print sentences using one random element from each cell array (e.g. 'Xavier eats sushi').

Ch8Ex5.m

```
% Create random silly sentences from cell arrays

names = {'Harry', 'Xavier', 'Sue'};
verbs = {'loves', 'eats'};
nouns = {'baseballs', 'rocks', 'sushi'};

name = names{randi([length(names)])};
verb = verbs{randi([length(verbs)])};
noun = nouns{randi([length(nouns)])};

fprintf('%s %s %s\n', name, verb, noun)
```

6) Write a script that will prompt the user for strings and read them in, store them in a cell array (in a loop), and then print them out.

Ch8Ex6.m

```
% Prompt the user for strings and store in a cell array

for i = 1:4
    str = input('Enter a string: ','s');
    strcell{i} = str;
end
strcell
```

7) When would you loop through the elements of a cell array?

When you want to do something with every element, such as getting the length of each element.

8) Write a function *buildstr* that will receive a character and a positive integer n. It will create and return a cell array with strings of increasing lengths, from 1 to the integer *n*. It will build the strings with successive characters in the ASCII encoding.

```
>> buildstr('a',4)
ans =
    'a'    'ab'    'abc'    'abcd'
```

buildstr.m

```
function outcell = buildstr(inchar, posint)
% Creates a cell array with strings of increasing
% lengths, from 1:n, starting with inchar
% Format of call: buildstr(input char, n)
% Returns cell array with n strings

outcell =cell(1,posint);
inchar = char(inchar-1);

strin = '';

for i = 1:posint
    strin = strcat(strin, char(inchar+i));
    outcell{i} = strin;
end
end
```

9) Write a function “catit” that will receive one input argument which is a cell array. If the cell array contains only strings, it will return one

string which is all of the strings from the cell array concatenated together – otherwise, it will return an empty string. Here is one example of calling the function:

```
>> fishies = {'tuna','shark','salmon','cod'};
>> catit(fishies)
ans =
tunasharksalmoncod
>>
```

Do this using the programming method.

catit.m

```
function outsent = catit(ca)
outsent = '';
if iscellstr(ca)
   for i = 1:length(ca)
       outsent = strcat(outsent,ca{i});
   end
end
end
```

10) Modify the previous function to use the **strjoin** function. Is there a difference?

catitii.m

```
function outsent = catitii(ca)
outsent = '';
if iscellstr(ca)
   outsent = strjoin(ca);
end
end
```

Yes:

```
>> catitii(fishies)
ans =
tuna shark salmon cod
```

11) Create a cell array variable that would store for a student his or her name, university id number, and GPA. Print this information.

```
>> studentca= {'Smith, Susan', 12345678, 3.5};
>> fprintf('Name: %s\nUID: %d\nGPA: %.2f\n', studentca{1}, ...
      studentca{2}, studentca{3})
Name: Smith, Susan
UID: 12345678
GPA: 3.50
```

12) Create a structure variable that would store for a student his or her name, university id number, and GPA. Print this information.

```
>> studentst = struct('name', 'Smith, Susan', 'UID', ...
    12345678, 'GPA', 3.5);
>> fprintf('name: %s\nUID: %d\nGPA: %.2f\n', studentst.name,...
    studentst.UID, studentst.GPA)
name: Smith, Susan
UID: 12345678
GPA: 3.50
```

13) Here is an inefficient way of creating a structure variable to store a person's name as first, middle, and last:

```
>> myname.first = 'Homer';
>> myname.middle = 'James';
>> myname.last = 'Fisch';
```

Re-write this more efficiently using the **struct** function:

```
>> myname = struct('first','Homer','middle','James',...
    'last', 'Fisch')
```

14) What would be an advantage of using cell arrays over structures?

```
You can index into a cell array, so you can loop through the
elements (or, vectorize and have MATLAB do that for you).
```

15) What would be an advantage of using structures over cell arrays?

```
   The fields are named so they're more mnemonic.
```

16) A complex number is a number of the form a + ib, where a is called the real part, b is called the imaginary part, and i = $\sqrt{-1}$. Write a script that prompts the user separately to enter values for the real and imaginary parts, and stores them in a structure variable. It then prints the complex number in the form a + ib. The script should just print the value of a, then the string '+ i', and then the value of b. For example, if the script is named "compnumstruct", running it would result in:

```
>> compnumstruct
Enter the real part: 2.1
Enter the imaginary part: 3.3
The complex number is 2.1 + i3.3
```

compnumstruct.m

```
% Prompt the user separately for the real and
```

```
% imaginary parts of a complex number, store in
% a structure, and print it

compnum.real = input('Enter the real part: ');
compnum.imag = input('Enter the imaginary part: ');

fprintf('The complex number is %.1f +i%.1f\n', ...
  compnum.real, compnum.imag)
```

17) Create a data structure to store information about the elements in the periodic table of elements. For every element, store the name, atomic number, chemical symbol, class, atomic weight, and a seven-element vector for the number of electrons in each shell. Create a structure variable to store the information, for example for lithium:

```
     Lithium 3 Li alkali_metal 6.94 2 1 0 0 0 0 0

>> element = struct('name','Lithium','atomicNumber',3,...
'chemicalSymbol','Li','class','alkali_metal',...
'atomicWeight',6.94, 'electrons',[2 1 0 0 0 0 0])
element =
              name: 'Lithium'
      atomicNumber: 3
    chemicalSymbol: 'Li'
             class: 'alkali_metal'
      atomicWeight: 6.9400
         electrons: [2 1 0 0 0 0 0]
```

18) Write a function “separatethem” that will receive one input argument which is a structure containing fields named ‘length’ and ‘width’, and will return the two values separately. Here is an example of calling the function:

```
     >> myrectangle = struct('length',33,'width',2);
     >> [l w] = separatethem(myrectangle)
     l =
         33
     w =
          2
```

separatethem.m

```
function [len, wid] = separatethem(rect)
len = rect.length;
wid = rect.width;
end
```

19) In chemistry, the pH of an aqueous solution is a measure of its acidity. A solution with a pH of 7 is said to be *neutral*, a solution with a pH greater than 7 is *basic*, and a solution with a pH less than 7 is *acidic*. Create a vector of structures with various solutions and their pH values. Write a function that will determine acidity. Add another field to every structure for this.

Ch8Ex19.m

```
% Create a vector of structures to store solution
% names and their pH values

phvals(2) = struct('solname', 'water', 'ph', 7);
phvals(1) = struct('solname', 'coffee', 'ph', 5);

new = addAcidity(phvals)
```

addAcidity.m

```
function outph = addAcidity(phvals)
% Determines acidity for solutions and adds
% a field to the phvals structures
% Format of call: addAcidity(vec of solutions structs)
% Returns new vector of structs with acidity field

outph = phvals;

for i = 1:length(phvals)
    if phvals(i).ph == 7
        outph(i).acidity = 'neutral';
    elseif phvals(i).ph < 7
        outph(i).acidity = 'acidic';
    else
        outph(i).acidity = 'basic';
    end
end
end
```

20) A script stores information on potential subjects for an experiment in a vector of structures called *subjects*. The following shows an example of what the contents might be:

```
>> subjects(1)
ans =
      name: 'Joey'
    sub_id: 111
    height: 6.7000
    weight: 222.2000
```

For this particular experiment, the only subjects who are eligible are those whose height or weight is lower than the average height or weight of all subjects. The script will print the names of those who are eligible. Create a vector with sample data in a script, and then write the code to accomplish this. Don't assume that the length of the vector is known; the code should be general.

Ch8Ex20.m

```
% create vector of structures on potential subjects for an
% experiment, calculate ave height and weight, and
% determine and print eligible participants

subjects(3) = struct('name','Mary','sub_id',363,'height',5.1,...
'weight',110);
subjects(1) = struct('name','Joey','sub_id',111,'height',6.7,...
'weight',222.2);
subjects(2) = struct('name','Pat','sub_id',221,'height',5.9,...
'weight',165);

%calculate the average height and weight
avgheight = sum([subjects.height])/length(subjects);
avgweight = sum([subjects.weight])/length(subjects);


%find and print the eligible participants
disp('Eligible Subjects:')
for i = 1:length(subjects)
    if subjects(i).height < avgheight || subjects(i).weight <
avgweight
        fprintf('%s\n',subjects(i).name)
    end
end
```

21) Quiz data for a class is stored in a file. Each line in the file has the student ID number (which is an integer) followed by the quiz scores for that student. For example, if there are four students and three quizzes for each, the file might look like this:

```
44   7    7.5  8
33   5.5  6    6.5
37   8    8    8
24   6    7    8
```

First create the data file, and then store the data in a script in a vector of structures. Each element in the vector will be a structure that has 2 members: the integer student ID number, and a vector of quiz scores. The structure will look like this:

students

		quiz		
	id_no	1	2	3
1	44	7	7.5	8
2	33	5.5	6	6.5
3	37	8	8	8
4	24	6	7	8

To accomplish this, first use the **load** function to read all information from the file into a matrix. Then, using nested loops, copy the data into a vector of structures as specified. Then, the script will calculate and print the quiz average for each student.

Ch8Ex21.m

```
% Read student info from a file into a vector of structures
% and print each student's name and quiz average
load studentinfo.dat

[r c] = size(studentinfo);
for i = 1:r
   students(i).id_no = studentinfo(i,1);
   for j = 1:c-1
      students(i).quiz(j) = studentinfo(i,j+1);
   end
end

fprintf('  Student    Quiz Ave\n')
for i = 1:r
   fprintf('     %d          %.2f\n', students(i).id_no,
mean(students(i).quiz))
end
```

22) Create a nested struct to store a person's name, address, and phone numbers. The struct should have 3 fields for the name, address, and phone. The address fields and phone fields will be structs.

```
>> person = ...
 struct('name','Mary','address',struct('street',...
'226 West Elm Rd','City','Boston','State','MA',...
'ZipCode',02215),'PhoneNum',struct('AreaCode',...
617,'Number',9156687));
```

23) Design a nested structure to store information on constellations for a rocket design company. Each structure should store the

constellation's name and information on the stars in the constellation. The structure for the star information should include the star's name, core temperature, distance from the sun, and whether it is a binary star or not. Create variables and sample data for your data structure.

```
 constellations(4) = struct('name','Ursa Major',...
'stars',struct('name','Dubhe','CoreTemp',4500,...
'DistFromSun',124,'Binary', 'yes'));

constellations(3) = struct('name','Ursa Minor',...
'stars',struct('name','Polaris','CoreTemp',6000,...
'DistFromSun',430,'Binary', 'yes'));

constellations(2).stars(2) = struct('name',...
'Mizar','CoreTemp',9600,'DistFromSun',78,'Binary','yes');

constellations(1).stars(2) = struct('name',...
'Kochab','CoreTemp',4000,'DistFromSun',126,'Binary','no');
```

24) Write a script that creates a vector of line segments (where each is a nested structure as shown in this chapter). Initialize the vector using any method. Print a table showing the values, such as shown in the following:

```
Line From      To
==== =======   =======
  1  ( 3, 5)   ( 4, 7)
  2  ( 5, 6)   ( 2, 10)
     etc.
```

Ch8Ex24.m

```
% Create a vector of line segments and
% print a table showing the values

fprintf('Line From\t\tTo\n')
fprintf('==== =======\t=======\n')

for i = 1:10 % 10 segments
    linestr(i).from.x = randint(1,1,[-10,10]);
    linestr(i).from.y = randint(1,1,[-10,10]);
    linestr(i).to.x = randint(1,1,[-10,10]);
    linestr(i).to.y = randint(1,1,[-10,10]);
    fprintf('  %d  ( %d, %d)\t( %d, %d)\n',...
    i,linestr(i).from.x, linestr(i).from.y,...
    linestr(i).to.x, linestr(i).to.y)
end
```

25) Given a vector of structures defined by the following statements:

```
kit(2).sub.id = 123;
kit(2).sub.wt = 4.4;
kit(2).sub.code = 'a';
kit(2).name = 'xyz';
kit(2).lens = [4 7];
kit(1).name = 'rst';
kit(1).lens = 5:6;
kit(1).sub.id = 33;
kit(1).sub.wt = 11.11;
kit(1).sub.code = 'q';
```

Which of the following expressions are valid? If the expression is valid, give its value. If it is not valid, explain why.

```
>> kit(1).sub
ans =
      id: 33
      wt: 11.1100
    code: 'q'
>> kit(2).lens(1)
ans =
     4
>> kit(1).code
Reference to non-existent field 'code'.

>> kit(2).sub.id == kit(1).sub.id
ans =
     0
>> strfind(kit(1).name, 's')
ans =
     2
```

26) Create a vector of structures *experiments* that stores information on subjects used in an experiment. Each struct has four fields: *num*, *name*, *weights*, and *height*. The field *num* is an integer, *name* is a string, *weights* is a vector with two values (both of which are double values), and *height* is a struct with fields *feet* and *inches* (both of which are integers). The following is an example of what the format might look like.

	num	name	experiments weights		height	
			1	2	feet	inches
1	33	Joe	200.34	202.45	5	6

2	11	Sally	111.45	111.11	7	2

Write a function *printhts* that will receive a vector in this format and will print the name and height of each subject in inches (1 foot = 12 inches). This function calls another function *howhigh* that receives a height struct and returns the total height in inches. This function could also be called separately.

Ch8Ex26.m

```
% Create an "experiments" vector of structures
% variable, and pass it to a function that will
% print the height of each subject

experiments(2) = struct('num', 11, 'name', 'Sally', ...
    'weights', [111.45, 111.11], 'height',...
    struct('feet',7, 'inches',2));
experiments(1) = struct('num',33,'name', 'Joe', ...
    'weights', [200.34 202.45], 'height', ...
    struct('feet', 5, 'inches', 6));

printhts(experiments)
```

printhts.m

```
function printhts(exps)
% Prints height of every subject
% Format of call: printhts(experiments vector)
% Does not return any values

for i = 1: length(exps)
    high = howhigh(exps(i).height);
    fprintf('%s is %d inches tall\n', ...
        exps(i).name, high)
end
end
```

howhigh.m

```
function ht = howhigh(expstruct)
% Calculates height in inches of a subject
% Format of call: howhigh(experiment struct)
% Returns height in inches

ht = expstruct.feet*12+expstruct.inches;
end
```

27) A team of engineers is designing a bridge to span the Podunk River. As part of the design process, the local flooding data must be

analyzed. The following information on each storm that has been recorded in the last 40 years is stored in a file: a code for the location of the source of the data, the amount of rainfall (in inches), and the duration of the storm (in hours), in that order. For example, the file might look like this:

```
321   2.4   1.5
111   3.3   12.1
     etc.
```

Create a data file. Write the first part of the program: design a data structure to store the storm data from the file, and also the intensity of each storm. The intensity is the rainfall amount divided by the duration. Write a function to read the data from the file (use **load**), copy from the matrix into a vector of structs, and then calculate the intensities. Write another function to print all of the information in a neatly organized table.

Add a function to the program to calculate the average intensity of the storms.

Add a function to the program to print all of the information given on the most intense storm. Use a subfunction for this function which will return the index of the most intense storm.

flooding.m

```
% Process flood data

flooddata = floodInfo;
printflood(flooddata)
calcavg(flooddata)
printIntense(flooddata)
```

floodInfo.m

```
function flood = floodInfo
% load flood information and store in vector
% Format of call: floodInfo or floodInfo()
% Returns vector of structures

load floodData.dat
[r c] = size(floodData);

for i=1:r
    flood(i) = struct('source',floodData(i,1),'inches',...
        floodData(i,2),'duration',floodData(i,3),...
        'intensity', floodData(i,2)/floodData(i,3));
end
end
```

calcavg.m

```
function calcavg(flooddata)
% Calculates the ave storm intensity
% Format of call: calcavg(flooddata)
% Returns average storm intensity

avginten = sum([flooddata.intensity])/length(flooddata);
fprintf('The average intensity of the storms is %.4f',...
   avginten);
end
```

printflood.m

```
function printflood(flooddata)
% Prints flood info
% Format of call: printflood(flooddata)
% Does not return any values

for i = 1:length(flooddata)
    fprintf('Flood Source: %d\n')
    fprintf('Total Rainfall (in inches): %.2f\n')
    fprintf('Duration of Storm: %.2f\n')
    fprintf('Intensity: %.3f\n\n', ...
        flooddata(i).source, flooddata(i).inches,...
        flooddata(i).duration, flooddata(i).intensity)
end
end
```

printIntense.m

```
function printIntense(flooddata)
% Prints info on most intense storm
% Format of call: printIntense(flooddata)
% Does not return any values

ind = findind(flooddata);

fprintf('\nThe most intense recorded storm began')
fprintf(' flooding at location %d.\n')
fprintf('%.2f inches of rain fell in %.2f hours\n\n',...
   flooddata(ind).source, flooddata(ind).inches, ...
   flooddata(ind).duration)
end
```

findind.m

```
function ind = findind(flooddata)
% Determines most intense storm
% Format of call: findind(flooddata)
% Returns index of most intense storm
```

```
intensity = [flooddata.intensity];
mostintense = intensity(1);
ind = 1;

%search for the highest intensity value
for i=1:length(intensity)
    if intensity(i) > mostintense
       %if higher intensity is found, save value and index
        mostintense = intensity(i);
        ind = i;
    end
end
end
```

28) Create an ordinal categorical array to store the four seasons.

```
>> seasons = {'Summer', 'Fall', 'Winter', 'Spring'};
>> seasonset = {'Fall', 'Summer', 'Spring', 'Spring', ...
      'Winter', 'Summer'};
>> ordss = categorical(seasonset, seasons, 'Ordinal', true)
ordss =
     Fall     Summer     Spring     Spring     Winter    Summer
>> summary(ordss)
     Summer      Fall      Winter      Spring
          2         1           1           2
```

29) Create a table to store information on students; for each, their name, id number, and major.

```
>> names = {'Carlton', 'Raaid', 'Igor'};
>> ids = {'123'; '234'; '345'};
>> majors = {'CE'; 'EE'; 'CE'};
>> awesomestudents = table(ids, majors, 'RowNames', names)
awesomestudents =
                ids     majors
              _____    _______
    Carlton   '123'    'CE'
    Raaid     '234'    'EE'
    Igor      '345'    'CE'
```

30) Write a function *mydsort* that sorts a vector in descending order (using a loop, not the built-in sort function).

mydsort.m

```
function outv = mydsort(vec)
```

```
% This function sorts a vector using the selection sort
% Format of call: mydsort(vector)
% Returns the vector sorted in descending order

for i = 1:length(vec)-1
    highind = i;
    for j=i+1:length(vec)
        if vec(j) > vec(highind)
            highind = j;
        end
    end
    % Exchange elements
    temp = vec(i);
    vec(i) = vec(highind);
    vec(highind) = temp;
end
outv = vec;
end
```

31) Write a function *matsort* to sort all of the values in a matrix (decide whether the sorted values are stored by row or by column). It will receive one matrix argument and return a sorted matrix. Do this without loops, using the built-in functions **sort** and **reshape**. For example:

```
>> mat
mat =
     4     5     2
     1     3     6
     7     8     4
     9     1     5
>> matsort(mat)
ans =
     1     4     6
     1     4     7
     2     5     8
     3     5     9
```

matsort.m

```
function outmat = matsort(mat)
% Sorts ALL of the values in a matrix and
% then stores them column-wise
% Format of call: matsort(matrix)
% Returns a matrix, sorted and stored by columns

[r c] = size(mat);
```

```
vec = reshape(mat, 1, r*c);
vs = sort(vec);
outmat = reshape(vs,r,c);
end
```

32) DNA is a double stranded helical polymer that contains basic genetic information in the form of patterns of nucleotide bases. The patterns of the base molecules A, T, C, and G encode the genetic information. Construct a cell array to store some DNA sequences as strings; such as
TACGGCAT
ACCGTAC
and then sort these alphabetically. Next, construct a matrix to store some DNA sequences of the same length and then sort them alphabetically.

```
>> strings = {'TACGGCAT','ACCGTAC'};
>> sort(strings)
ans =
    'ACCGTAC'    'TACGGCAT'

>> mat = ['TACCGGCAT';'ACCGTACGT'];
>> sortrows(mat)
ans =
ACCGTACGT
TACCGGCAT
```

33) Trace this; what will it print?

parts

	code	quantity	weight
1	'x'	11	4.5
2	'z'	33	3.6
3	'a'	25	4.1
4	'y'	31	2.2

	ci
1	3
2	1
3	4
4	2

	qi
1	1
2	3
3	4
4	2

	wi
1	4
2	2
3	3
4	1

```
for i = 1:length(parts)
   fprintf('Part %c weight is %.1f\n',...
       parts(qi(i)).code, parts(qi(i)).weight)
end

Part x weight is 4.5
Part a weight is 4.1
Part y weight is 2.2
Part z weight is 3.6
```

34) When would you use sorting vs. indexing?

Indexing: when you have a vector of structures and you want to get it in order of different fields, and/or if you want to leave the vector in its original order
Sorting: When you have one vector and don't need the original order

35) Write a function that will receive a vector and will return two index vectors: one for ascending order and one for descending order. Check the function by writing a script that will call the function and then use the index vectors to print the original vector in ascending and descending order.

Ch8Ex35.m

```
% Test the createinds function by creating
% a test vector and using index vectors in
% ascending and descending order

vec = [5 99 22 1 6 0 -4 33];
[a, d] = createinds(vec);
vec(a)
vec(d)
```

createinds.m

```
function [ascind, desind] = createinds(vec)
% This function creates two index vectors:
% in ascending order and descending order
% Format of call: createinds(vector)
% Returns index vector in ascending order and then
%  another index vector in descending order

% Ascending
% Initialize the index vector
len = length(vec);
ascind = 1:len;

for i = 1:len-1
    low = i;
    for j=i+1:len
        % Compare values in the original vector
        if vec(ascind(j)) < vec(ascind(low))
            low = j;
        end
     end
```

```
    % Exchange elements in the index vector
    temp = ascind(i);
    ascind(i) = ascind(low);
    ascind(low) = temp;
end

 % Descending
 % Could of course just reverse the ascind vector
 % e.g. desind = fliplr(ascind);
 % Programming method: Initialize the index vector
len = length(vec);
desind = 1:len;

for i = 1:len-1
    high = i;
    for j=i+1:len
        % Compare values in the original vector
        if vec(desind(j)) > vec(desind(high))
            high = j;
        end
     end
    % Exchange elements in the index vector
    temp = desind(i);
    desind(i) = desind(high);
    desind(high) = temp;
end
end
```

Chapter 9: Advanced File Input and Output

Exercises

1) Create a spreadsheet that has on each line an integer student identification number followed by three quiz grades for that student. Read that information from the spreadsheet into a matrix, and print the average quiz score for each student.

Ch9Ex1.m

```
% Read student data from a spreadsheet and print
% quiz averages
mat = xlsread('quiz.xlsx'); %Read in data from a spreadsheet
[r, c] = size(mat);
for i = 1:r
    quizave = sum(mat(i,2:end))/3; %Calculate quiz average
    fprintf('Student #%d has a quiz average of %.1f\n',...
```

```
        mat(i,1),quizave)
end
```

2) The **xlswrite** function can write the contents of a cell array to a spreadsheet. A manufacturer stores information on the weights of some parts in a cell array. Each row stores the part identifier code followed by weights of some sample parts. To simulate this, create the following cell array:

```
>> parts = {'A22', 4.41 4.44 4.39 4.39
            'Z29', 8.88 8.95 8.84 8.92}
```

Then, write this to a spreadsheet file.

```
>> xlswrite('parts.xls', parts);
```

3) A spreadsheet *popdata.xlsx* stores the population every 20 years for a small town that underwent a boom and then decline. Create this spreadsheet (include the header row) and then read the headers into a cell array and the numbers into a matrix. Plot the data using the header strings on the axis labels.

Year	Population
1920	4021
1940	8053
1960	14994
1980	9942
2000	3385

Ch9Ex3.m

```
% Read population data from a spreadsheet
% and plot it

%Read numbers and text into separate variables
[num, txt] = xlsread('popdata.xlsx');

%Create vectors for the plot
year = num(:,1);
pop = num(:,2);
plot(year,pop','ko')

%Axes labels based on header strings
xlabel(txt{1})
ylabel(txt{2})
```

4) Create a multiplication table and write it to a spreadsheet.

Ch9Ex4.m

```
% Write a multiplication table to a spreadsheet

%Initialize multiplication table
mat = zeros(101,101);

mat(1,2:end) = 1:100; %Row header
mat(2:end,1) = 1:100; %Column header

%Create multiplication table
for i = 1:100
    for j = 1:100
        mat(i+1,j+1) = i*j;
    end
end

xlswrite('multable.xls', mat)
```

5) Read numbers from any spreadsheet file, and write the variable to a MAT-file.

```
>> mat = xlsread('randnum.xls');
>> save randnum mat
```

6) Clear out any variables that you have in your Command Window. Create a matrix variable and two vector variables.

- Make sure that you have your Current Folder set.
- Store all variables to a MAT-file
- Store just the two vector variables in a different MAT-file
- Verify the contents of your files using **who**.

```
>> mat = [2:4; 6:-1:4];
>> vec1 = 1:5;
>> vec2 = 4:12;
>> save matfile1
>> save matfile2 vec1 vec2
>> clear
>> load matfile1
>> who

Your variables are:

mat   vec1  vec2

>> clear
>> load matfile2
>> who
```

```
Your variables are:

vec1  vec2
```

7) Create a set of random matrix variables with descriptive names (e.g. *ran2by2int*, *ran3by3double*, etc.) for use when testing matrix functions. Store all of these in a MAT-file.

```
>> clear
>> ran2by2int = randi([0,100], 2,2);
>> ran3by3int = randi([0,100], 3,3);
>> ran2by2double = rand(2,2)*100;
>> ran3by3double = rand(3,3)*100;
>> save randtestmat
>> load randtestmat
>> ran2by2int
ran2by2int =
    71    27
     3     4
```

8) What is the difference between a data file and a MAT-file?

```
A data file just stores data in a text, or ASCII, format.
MAT-files store variables: their names AND their values.
```

9) Write a script that will prompt the user for the name of a file from which to read. Loop to error-check until the user enters a valid filename that can be opened. (Note: this would be part of a longer program that would actually do something with the file, but for this problem all you have to do is to error-check until the user enters a valid filename that can be read from.)

Ch9Ex9.m

```
% Prompt the user for a file name and error-check
% until the user enters a valid file name

fname = input('Enter a file name: ','s');
fid = fopen(fname);
while fid == -1
    fname = input('Error! Enter a valid file name: ', 's');
end
%Error-check file close
closeresult = fclose(fid);
if closeresult == 0
    disp('File close successful')
```

```
else
    disp('File close not successful')
end
```

10) A file “potfilenames.dat” stores potential file names, one per line. The names do not have any extension. Write a script that will print the names of the valid files, once the extension “.dat” has been added. “Valid” means that the file exists in the Current Directory, so it could be opened for reading. The script will also print how many of the file names were valid.

Ch9Ex10.m

```
fid = fopen('potfilenames.dat');
if fid == -1
    disp('File not opened')
else
    count = 0;
    while ~feof(fid)
        aname = fgetl(fid);
        aname = strcat(aname,'.dat');
        tryit = fopen(aname);
        if tryit ~= -1
            count = count + 1;
            fprintf('%s is a valid file\n', aname)
            fc = fclose(tryit); % Assume this works
        end
    end
    closeres = fclose(fid); % Assume this works
    fprintf('\nThere were %d valid file names\n', count)
end
```

11) A set of data files named “exfile1.dat”, “exfile2.dat”, etc. has been created by a series of experiments. It is not known exactly how many there are, but the files are numbered sequentially with integers beginning with 1. The files all store combinations of numbers and characters, and are not in the same format. Write a script that will count how many lines total are in the files. Note that you do not have to process the data in the files in any way; just count the number of lines.

Ch9Ex11.m

```
sumlines = 0;
i = 1;
filename = 'exfile1.dat';
fid = fopen(filename);
```

```
while fid ~= -1
    while ~feof(fid)
        aline = fgetl(fid);
        sumlines = sumlines + 1;
    end
    i = i + 1;
    filename = sprintf('exfile%d.dat', i);
    fid = fopen(filename);
end
fprintf('There were %d lines in the files\n', sumlines)
fclose('all');
```

12) Write a script that will read from a file x and y data points in the following format:

```
 x 0 y 1
 x 1.3 y 2.2
```

The format of every line in the file is the letter 'x', a space, the x value, space, the letter 'y', space, and the y value. First, create the data file with 10 lines in this format. Do this by using the Editor/Debugger, then File Save As *xypoints.dat*. The script will attempt to open the data file and error-check to make sure it was opened. If so, it uses a **for** loop and **fgetl** to read each line as a string. In the loop, it creates x and y vectors for the data points. After the loop, it plots these points and attempts to close the file. The script should print whether or not the file was successfully closed.

Ch9Ex12.m

```
%Read data points from "xypoints.dat" and plot them

fid = fopen('xypoints.dat');
if fid == -1
    disp('File open not successful')
else
    %Initialize x vector and y vector
    xvec = 1:10;
    yvec = 1:10;

    for i = 1:10
        aline = fgetl(fid);
        %Separate each line into two parts: x and y
        [x, rest] = strtok(aline,'y');
        x(1:2) = []; %Removes the "x" and the space
        [let, y] = strtok(rest);
        xvec(i) = str2num(x);
        yvec(i) = str2num(y);
    end
```

```
    plot(xvec,yvec,'ko')
    xlabel('x')
    ylabel('y')
    title('Points from file "xypoints.dat"')

    %Error-check file close
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

13) Modify the script from the previous problem. Assume that the data file is in exactly that format, but do not assume that the number of lines in the file is known. Instead of using a **for** loop, loop until the end of the file is reached. The number of points, however, should be in the plot title.

Ch9Ex13.m

```
%Read data points from "xypts.dat" and plot

fid = fopen('xypts.dat');
if fid == -1
    disp('File open not successful')
else
    %Initialize counter variable
    i = 0;
    while feof(fid) == 0
        aline = fgetl(fid);
        %Separate each line into two parts: x and y
        [x, rest] = strtok(aline,'y');
        x(1:2) = []; %Removes the "x" and the space
        [let, y] = strtok(rest);
        i = i + 1;
        xvec(i) = str2num(x);
        yvec(i) = str2num(y);
    end
    plot(xvec,yvec,'ko')
    xlabel('x')
    ylabel('y')
    title(sprintf('Number of data points: %d',i))

    %Error-check file close
    closeresult = fclose(fid);
```

```
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

Medical organizations store a lot of very personal information on their patients. There is an acute need for improved methods of storing, sharing, and encrypting all of these medical records. Being able to read from and write to the data files is just the first step.

14) For a biomedical experiment, the names and weights of some patients have been stored in a file *patwts.dat*. For example, the file might look like this:

```
Darby George  166.2
Helen Dee  143.5
Giovanni Lupa  192.4
Cat Donovan  215.1
```

Create this data file first. Then, write a script *readpatwts* that will first attempt to open the file. If the file open is not successful, an error message should be printed. If it is successful, the script will read the data into strings, one line at a time. Print for each person the name in the form 'last,first' followed by the weight. Also, calculate and print the average weight. Finally, print whether or not the file close was successful. For example, the result of running the script would look like this:

```
>> readpatwts
George,Darby   166.2
Dee,Helen   143.5
Lupa,Giovanni   192.4
Donovan,Cat   215.1
The ave weight is 179.30
File close successful
```

Ch9Ex14.m

```
% Reads names and weights of patients for an experiment
% Prints in form last, first and then weight
% Calculates and prints average patient weight

fid = fopen('patwts.dat');
if fid == -1
    disp('File open not successful')
else
   i = 0;
   % Store all weights in a vector
```

```
    weight = [];
    while feof(fid) == 0
        % Get first name, last name, and weight
        aline = fgetl(fid);
        [fname, rest] = strtok(aline);
        [lname, strwt] = strtok(rest);
        weight = [weight; str2num(strwt)];
        fprintf('%s,%s %s\n',lname,fname,strwt)
    end
    closeresult = fclose(fid);

    fprintf('The ave weight is %.2f\n',mean(weight))

    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

15) Create a data file to store blood donor information for a biomedical research company. For every donor, store the person's name, blood type, Rh factor, and blood pressure information. The Blood type is either A, B, AB, or O. The Rh factor is + or -. The blood pressure consists of two readings: systolic and diastolic (both are **double** numbers). Write a script to read from your file into a data structure and print the information from the file.

Ch9Ex15.m

```
% Reads blood donor information and store in vector
% of structures, print it out

fid = fopen('blooddonors.dat');
if fid == -1
    disp('File open not successful')
else
    i = 0;
    while feof(fid) == 0
        % Get first name, last name, blood type,
        % Rh factor, systolic & diastolic
        i = i + 1;
        aline = fgetl(fid);
        [fname, rest] = strtok(aline);
        [lname, rest] = strtok(rest);
        [bloodtype, rest] = strtok(rest);
        [rh, rest] = strtok(rest);
```

```
        [sys, dias] = strtok(rest);
        dstr = struct('First', fname, 'Last', lname, ...
            'BType', bloodtype, 'Rh', rh, ...
            'BPressure',struct('Systolic', str2num(sys), ...
            'Diastolic', str2num(dias)));
        donors(i) = dstr;
    end
    for i = 1:length(donors)
        fprintf('%-12s %-12s %4s', donors(i).Last, ...
            donors(i).First, donors(i).BType)
        fprintf('%2s %7.2f %7.2f\n', donors(i).Rh, ...
            donors(i).BPressure.Systolic, ...
            donors(i).BPressure.Diastolic)
    end

    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

16) A data file called “mathfile.dat” stores three characters on each line: an operand (a single digit number), an operator (a one character operator, such as `+, -, /, \, *, ^`), and then another operand (a single digit number). For example, it might look like this:

```
>> type mathfile.dat
5+2
8-1
3+3
```

You are to write a script that will use **fgetl** to read from the file, one line at a time, perform the specified operation, and print the result.

Ch9Ex16.m

```
% Read in math operations from a data file,
%  perform the operation and print the result
fid = fopen('mathfile.dat');
if fid == -1
    disp('File open not successful')
else
    while feof(fid) == 0
        % Get operand, operator, operand
        % and perform the operation!
        aline = fgetl(fid);
        res = eval(aline);
```

```
        fprintf('%s = %d\n', aline, res)
    end
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

17) Assume that a file named *testread.dat* stores the following:

```
110x0.123y5.67z8.45
120x0.543y6.77z11.56
```

Assume that the following are typed SEQUENTIALLY. What would the values be?

```
tstid = fopen('testread.dat')

fileline = fgetl(tstid)

[beg, endline] = strtok(fileline,'y')

length(beg)

feof(tstid)
```

```
>> tstid = fopen('testread.dat')
tstid =
  %some integer value, depending on how many files are open
>> fileline = fgetl(tstid)
fileline =
110x0.123y5.67z8.45
>> [beg, endline] = strtok(fileline,'y')
beg =
110x0.123
endline =
y5.67z8.45
>> length(beg)
ans =
     9
>> feof(tstid)
ans =
     0
```

18) Create a data file to store information on hurricanes. Each line in the file should have the name of the hurricane, its speed in miles per hour, and the diameter of its eye in miles. Then, write a script to read this information from the file and create a vector of structures to store it. Print the name and area of the eye for each hurricane.

Ch9Ex18.m

```
% Reads hurricane information and store in vector
% of structures, print name and area for each

fid = fopen('hurricane.dat');
if fid == -1
    disp('File open not successful')
else
   i = 0;
   while feof(fid) == 0
       i = i + 1;
       aline = fgetl(fid);
       [hname, rest] = strtok(aline);
       [speed, diam] = strtok(rest);
       hstruc = struct('Name', hname, 'Speed', ...
           str2num(speed), 'Diam', str2num(diam));
       hurricane(i) = hstruc;
   end
   for i = 1:length(hurricane)
       fprintf('%s had area %.2f\n', hurricane(i).Name, ...
           pi * (hurricane(i).Diam/2)^2)
   end

   closeresult = fclose(fid);
   if closeresult == 0
       disp('File close successful')
   else
       disp('File close not successful')
   end
end
```

19) Create a file “parts_inv.dat” that stores on each line a part number, cost, and quantity in inventory, in the following format:

```
123 5.99 52
```

Use **fscanf** to read this information, and print the total dollar amount of inventory (the sum of the cost multiplied by the quantity for each part).

Ch9Ex19.m

```
% Read in parts inventory information from file
```

```
% using fscanf, and print total $ amount in inventory

fid = fopen('parts_inv.dat');
if fid == -1
    disp('File open not successful')
else
    % Read in data from file
    data = fscanf(fid,'%d %f %d',[3,inf]);
    cost = data(2,:);
    quantity = data(3,:);
    total = sum(cost .* quantity);
    fprintf('The total amount of inventory is $%.2f\n',...
         total)

    % Error-check file close
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

20) Students from a class took an exam for which there were 2 versions, marked either A or B on the front cover (½ of the students had version A, ½ Version B). The exam results are stored in a file called “exams.dat”, which has on each line the version of the exam (the letter ‘A’ or ‘B’) followed by a space followed by the integer exam grade. Write a script that will read this information from the file using **fscanf**, and separate the exam scores into two separate vectors: one for Version A, and one for Version B. Then, the grades from the vectors will be printed in the following format (using **disp**).

```
A exam grades:
    99    80    76

B exam grades:
    85    82   100
```

Note: no loops or selection statements are necessary!

Ch9Ex20.m

```
fid = fopen('exams.dat');
mat = fscanf(fid,'%c %d\n', [2 inf]);
av = char(mat(1,:)) == 'A';
as = mat(2,av);
bs = mat(2,~av);
```

```
disp('A exam grades:')
disp(as)
disp('B exam grades:')
disp(bs)
check = fclose(fid);
```

21) Create a file which stores on each line a letter, a space, and a real number. For example, it might look like this:

```
e 5.4
f 3.3
c 2.2
```

Write a script that uses **textscan** to read from this file. It will print the sum of the numbers in the file. The script should error-check the file open and close, and print error messages as necessary.

Ch9Ex21.m

```
% use textscan to read in from a file in the
% format letter number and sum the numbers

fid = fopen('letnum.dat');
if fid == -1
    disp('File open not successful')
else
    %Read in data from file
    C = textscan(fid,'%c %f');
    total = sum(C{2}); %Sum the numbers in the file
    fprintf('The sum of the numbers is %.2f\n',total)

    %Error-check file close
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

22) Write a script to read in division codes and sales for a company from a file that has the following format:

A 4.2
B 3.9

Print the division with the highest sales.

Ch9Ex22.m

```
% Read in company data and print the division with the
% highest sales
```

```
fid = fopen('sales.dat');
if fid == -1
    disp('File open not successful')
else
    %Read in data from file
    C = textscan(fid,'%c %f');
    [sales, index] = max(C{2}); %Find the max sales and its index
    code = C{1}(index);
    fprintf('Division %s has the highest sales of %.1f\n',...
       code,sales)

    %Error-check file close
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

23) A data file is created as a **char** matrix and then saved to a file; for example,

```
>> cmat = char('hello', 'ciao', 'goodbye')
cmat =
hello
ciao
goodbye
>> save stringsfile.dat cmat -ascii
```

Can the **load** function be used to read this in? What about **textscan**?

Yes (but will store ASCII equivalents). Yes, as strings.

24) Create a file of strings as in the previous exercise, but create the file by opening a new M-file, type in strings, and then save it as a data file. Can the **load** function be used to read this in? What about **textscan**?

No. Yes.

25) Write a script that creates a cell array of strings, each of which is a two-word phrase. The script is to write the first word of each phrase to a file “exitstrings.dat” in the format shown below. You do not have to error-check on the file open or file close. The script should be general and should work for any cell array containing two-word phrases.

Ch9Ex25.m

```
strca = {'hi there', 'hello all', 'I did'};
outfid = fopen('exitstrings.dat','w');
for i = 1:length(strca)
   [f r] = strtok(strca{i});
   fprintf(outfid, 'Word %d: %s\n', i, f);
end
fclose(outfid);
```

26) 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 follows:

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

Create a table showing WCFs for temperatures ranging from -20 to 55 in steps of 5, and wind speeds ranging from 0 to 55 in steps of 5. Write this to a file *wcftable.dat*. If you have version R2016a or later, write the script as a live script.

Ch9Ex26.m

```
% Write a table of Wind Chill Factors to a file

%Initialize temperature and wind speed vectors
T = -20:5:55;
V = 0:5:55;

%Initialize WFC table
table = zeros(length(T)+1,length(V)+1);
table(1,2:end) = V; %Row header of the table
table(2:end,1) = T; %Column header of the table

%Create the WFC table
for i = 1:length(T)
    for j = 1:length(V)
        WCF = 35.7 + 0.6*T(i) - 35.7*(V(j)^0.16) + ...
             0.43*T(i)*(V(j)^0.16);
        table(i+1,j+1) = WCF;
    end
end

%Save resulting matrix to a .dat file
save wcftable.dat table -ascii
```

27) Write a script that will loop to prompt the user for *n* circle radii. The script will call a function to calculate the area of each circle, and will write the results in sentence form to a file.

Ch9Ex27.m

```
% Write the areas of circles to a file
% Prompt the user for the radii

n = 5;

%Open a new file for writing
fid = fopen('circles.dat','w');
for i = 1:n
    radius = input('Enter the radius of a circle: ');
    while radius <= 0 %Error-check user input
        radius = input('Enter the radius of a circle: ');
    end
    area = calcarea(radius);

    %Write to file "circles.dat"
    fprintf(fid,'The area of circle #%d is %.2f\n',i,area);
end

%Error-check file close
closeresult = fclose(fid);
if closeresult == 0
    disp('File close successful')
else
    disp('File close not successful')
end
```

28) Create a file that has some college department names and enrollments. For example, it might look like this:

```
Aerospace 201
Mechanical 66
```

Write a script that will read the information from this file and create a new file that has just the first four characters from the department names, followed by the enrollments. The new file will be in this form:

```
Aero 201
Mech 66
```

Ch9Ex28.m

```
% Read in department names and enrollments from one file and
% write first 4 chars of names w enrollments to another file

fid = fopen('eng.dat');
if fid == -1
    disp('File open not successful')
else
    %Open a new file for writing
```

```
    nfid = fopen('neweng.dat','w');
    while feof(fid) == 0
        aline = fgetl(fid);
        %Seprate department names and enrollment
        [dep num] = strtok(aline);
        num = str2num(num);
        fprintf(nfid,'%s %d\n',dep(1:4),num); %Write to file
    end

    %Error-check file close
    closeresult = fclose('all');
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

29) An engineering corporation has a data file “vendorcust.dat” which has names of its vendors and customers for various products, along with a title line. The format is that every line has the vendor name and then the customer name, separated by one space. For example, it might look like this (although you cannot assume the length):

```
>> type vendorcust.dat

Vendor Customer
Acme XYZ
Tulip2you Flowers4me
Flowers4me Acme
XYZ Cartesian
```

The “Acme” company wants a little more zing in their name, however, so they’ve changed it to “Zowie”; now this data file has to be modified. Write a script that will read in from the “vendorcust.dat” file and replace all occurrences of “Acme” with “Zowie”, writing this to a new file called “newvc.dat”.

Ch9Ex29.m

```
fidin = fopen('vendorcust.dat');
fidout = fopen('newvc.dat','w');
while ~feof(fidin)
    aline = fgetl(fidin);
    newline = strrep(aline,'Acme','Zowie');
    fprintf(fidout,'%s\n',newline);
end
fclose('all');
```

30) Environmental engineers are trying to determine whether the underground aquifers in a region are being drained by a new spring water company in the area. Well depth data has been collected every year at several locations in the area. Create a data file that stores on each line the year, an alphanumeric code representing the location, and the measured well depth that year. Write a script that will read the data from the file and determine whether or not the average well depth has been lowered.

Ch9Ex30.m

```
% Read in well depth data for the last few years to
% determine whether the average depth has lowered or not

fid = fopen('wells.dat');
if fid == -1
    disp('File open not successful')
else
    aline = fgetl(fid);
    aveyear1 = avedepth(aline);
    while ~feof(fid)
        aline = fgetl(fid);
        aveyearx = avedepth(aline);
    end
    if aveyearx < aveyear1
        disp('The average well depth was lowered')
    else
        disp('The average well depth was not lowered')
    end

    %Error-check file close
    closeresult = fclose(fid);
    if closeresult == 0
        disp('File close successful')
    else
        disp('File close not successful')
    end
end
```

avedepth.m

```
function outave = avedepth(aline)
% Calculates the average well depth for a year
% Format of call: avedepth(a line from file)
% Returns the average well depth

[year, rest] = strtok(aline);
[code, rest] = strtok(rest);
```

```
[depth1, rest] = strtok(rest);
[code, rest] = strtok(rest);
[depth2, rest] = strtok(rest);
[code, depth3] = strtok(rest);
outave = mean([str2num(depth1) str2num(depth2) str2num(depth3)]);
end
```

31) Write a menu-driven program that will read in an employee data base for a company from a file, and do specified operations on the data. The file stores the following information for each employee:

- Name
- Department
- Birth Date
- Date Hired
- Annual Salary
- Office Phone Extension

You are to decide exactly how this information is to be stored in the file. Design the layout of the file, and then create a sample data file in this format to use when testing your program. The format of the file is up to you. However, space is critical. Do not use any more characters in your file than you have to! Your program is to read the information from the file into a data structure, and then display a menu of options for operations to be done on the data. You may not assume in your program that you know the length of the data file. The menu options are:

1. Print all of the information in an easy-to-read format to a new file.

2. Print the information for a particular department.

3. Calculate the total payroll for the company (the sum of the salaries).

4. Find out how many employees have been with the company for N years (N might be 10, for example).

5. Exit the program.

Ch9Ex31.m

```
% This script creates an employee data base for a company
%  and performs some operations on the data

% Read the info from a file
employees = reademployees;
% Call a function to display a menu and get choice
choice = options;
```

```
while choice ~= 5
   switch choice
      case 1
          % Prints all of the info to a file
          printall(employees)
      case 2
          % Prints info for one department
          printdept(employees)
      case 3
          % Prints total payroll
          payroll([employees.salary])
       case 4
          % Prints employees >= N years
          nyears(employees)
   end
   % Display menu again and get user's choice
   choice = options;
end
```

reademployees.m

```
function emp = reademployees
% Function stub
emp(2).name = 'Smith, Jane';
emp(2).dept = 'Service';
emp(2).birth = '072267';
emp(2).hired = '121298';
emp(2).salary = 87333;
emp(2).phone = '5388';
emp(1).name = 'Smith, Joe';
emp(1).dept = 'Sales';
emp(1).birth = '072267';
emp(1).hired = '121288';
emp(1).salary = 77333;
emp(1).phone = '5389';
end
```

options.m

```
function choice = options
% options prints the menu of options and error-checks
% until the user pushes one of the buttons

choice = menu('Choose an option', 'Print all', ...
    'Print dept', 'Payroll', 'N years', 'Exit Program');
% If the user closes the menu box rather than
% pushing one of the buttons, choice will be 0
while choice == 0
```

```
disp('Error-please choose one of the options.')
choice = menu('Choose an option', 'Print all', ...
    'Print dept', 'Payroll', 'N years', 'Exit Program');
end
end
```

printall.m

```
function printall(emp)
% Write to screen; could change to write to file
fprintf('%-15s%-8s%11s%11s  %-10s %5s\n\n', 'Name', 'Dept', ...
    'Birth Date', 'Hire Date', 'Salary', 'Phone')
for i = 1:length(emp)
    fprintf('%-15s%-8s', emp(i).name, emp(i).dept)
    b = emp(i).birth;
    birthdate = sprintf('%s-%s-19%s',b(1:2),b(3:4),b(5:6));
    h = emp(i).hired;
    hiredate = sprintf('%s-%s-19%s',h(1:2),h(3:4),h(5:6));
    fprintf('%11s%11s', birthdate, hiredate)
    fprintf('  $%9.2f x%s\n', emp(i).salary, emp(i).phone)
end

end
```

printdept.m

```
function printdept(emp)

choice = menu('Choose Dept', 'Sales', ...
    'Service', 'Trucking');
% If the user closes the menu box rather than
% pushing one of the buttons, choice will be 0
while choice == 0
   disp('Error-please choose one of the options.')
   choice = menu('Choose Dept', 'Sales', ...
    'Service', 'Trucking');
end

ca = {'Sales','Service','Trucking'};
chosen = ca{choice};

fprintf('%-15s%-8s%11s%11s  %-10s %5s\n\n', 'Name', 'Dept', ...
    'Birth Date', 'Hire Date', 'Salary', 'Phone')
for i = 1:length(emp)
    if strcmp(emp(i).dept, chosen)
       fprintf('%-15s%-8s', emp(i).name, emp(i).dept)
       b = emp(i).birth;
       birthdate = sprintf('%s-%s-19%s',b(1:2),b(3:4),b(5:6));
```

```
        h = emp(i).hired;
        hiredate = sprintf('%s-%s-19%s',h(1:2),h(3:4),h(5:6));
        fprintf('%11s%11s', birthdate, hiredate)
        fprintf('  $%9.2f x%s\n', emp(i).salary, emp(i).phone)
    end
end

end
```

payroll.m

```
function payroll(salaries)
fprintf('The total of the salaries is $%.2f.\n\n', ...
    sum(salaries))
end
```

nyears.m

```
function nyears(emps)
% Only considers years

hiredyears = zeros(1, length(emps));

for i = 1:length(emps)
    hiredyears(i) = str2num(emps(i).hired(5:6)) + 1900;
end
current_year = 2013;
N = 20;
fy = find(current_year-hiredyears >= N);

fprintf('%d employees have worked at least %d years.\n\n', ...
    length(fy), N)
end
```

Chapter 10: Advanced Functions

Exercises

1) Write a function that will print a random integer. If no arguments are passed to the function, it will print an integer in the inclusive range from 1 to 100. If one argument is passed, it is the max and the integer will be in the inclusive range from 1 to max. If two arguments are passed, they represent the min and max and it will print an integer in the inclusive range from min to max.

prtran.m

```
function prtran(varargin)
% Prints a random integer
% If no arguments are passed, the range is random
% otherwise the input argument(s) specify the range
% Format of call: prtran or prtran(maximum)
%   or prtran(minimum, maximum)
% Does not return any values

n = nargin;  % number of input arguments

% If no argument passed, range is 1-100
if n == 0
    myran = randi([1, 100]);
elseif n == 1
    % one argument is max, range is 1-max
    myran = randi([1 varargin{1}]);
elseif n == 2
    % two arguments are min-max
    myran = randi([varargin{1} varargin{2}]);
end

fprintf('The random integer is %d\n', myran)
end
```

2) Write a function *numbers* that will create a matrix in which every element stores the same number num. Either two or three arguments will be passed to the function. The first argument will always be the number num. If there are two arguments, the second will be the size of the resulting square (n x n) matrix. If there are three arguments, the second and third will be the number of rows and columns of the resulting matrix.

numbers.m

```
function outmat = numbers(num,varargin)
if nargin == 2
    outmat = ones(varargin{1}) * num;
else
    outmat = ones(varargin{1},varargin{2}) * num;
end
end
```

3) The overall electrical resistance of *n* resistors in parallel is given as:

$$R_T = \left(\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \ldots + \frac{1}{R_n} \right)^{-1}$$

Write a function *Req* that will receive a variable number of resistance values and will return the equivalent electrical resistance of the resistor network.

Req.m

```
function resis = Req(varargin)
% Calculates the resistance of n resistors in parallel
% Format of call: Req(res1, res2, ... , resn)
% Returns the overall resistance

resis = 0;
for i = 1:nargin
    resis = resis + 1/varargin{i};
end
resis = resis ^ -1;
end
```

4) Write a function that will receive the radius *r* of a sphere. It will calculate and return the volume of the sphere ($4/3\ \pi\ r^3$). If the function call expects two output arguments, the function will also return the surface area of the sphere ($4\ \pi\ r^2$).

spherecalcs.m

```
function [vol, varargout] = spherecalcs(r)
% Calculates and returns the volume of a sphere
% and possibly also the surface area
% Format of call: spherecalcs(radius)
% Returns volume of sphere, and if two output
% arguments are expected, also the surface area

vol = 4/3*pi*r^3;

if nargout == 2
    varargout{1} = 4*pi*r^2;
end
end
```

5) Most lap swimming pools have lanes that are either 25 yards long or 25 meters long; there's not much of a difference. A function "convyards" is to be written to help swimmers calculate how far they swam. The function receives as input the number of yards. It calculates and returns the equivalent number of meters, and, if (and only if) two output arguments are expected, it also returns the equivalent number of miles. The relevant conversion factors are:

```
1 meter = 1.0936133 yards
1 mile = 1760 yards
```

convyards.m

```
function [meters, varargout] = convyards(yards)
meters = yards / 1.0936133;
if nargout == 2
    varargout{1} = yards / 1760;
end
end
```

6) Write a function *unwind* that will receive a matrix as an input argument. It will return a row vector created columnwise from the elements in the matrix. If the number of expected output arguments is two, it will also return this as a column vector.

unwind.m

```
function [rowvec, varargout] = unwind(mat)
% Returns a row vector created columnwise from the
% input matrix, and possibly also a column vector
% Format of call: unwind(matrix)
% Returns a row vector from the matrix and if two
%  output arguments are expected, also a column vector

rowvec = mat(1:end);

if nargout == 2
    varargout{1} = rowvec';
end
end
```

7) Write a function "cylcalcs" that will receive the radius and height of a cylinder and will return the area and volume of the cylinder. If the function is called as an expression or in an assignment statement with one variable on the left, it will return the area and volume together in one vector. On the other hand, if the function is called in an assignment statement with two variables on the left, the function will return the area and volume separately (in that order). The function will call a subfunction to calculate the area. The formulas are:

$$Area = pi * radius^2$$
$$Volume = Area * height$$

cylcalcs.m

```
function [varargout] = cylcalcs(r,h)
a = area(r);
v = a*h;
if nargout <= 1
```

```
    varargout{1} = [a v];
else
    varargout{1} = a;
    varargout{2} = v;
end
end

function a = area(r)
a = pi * r ^ 2;
end
```

8) Information on some hurricanes is stored in a vector of structures; the name of the vector variable is *hurricanes*. For example, one of the structures might be initialized as follows:

```
     struct('Name','Bettylou', 'Avespeed',18,...
            'Size', struct('Width',333,'Eyewidth',22));
```

Write a function *printHurr* that will receive a vector of structures in this format as an input argument. It will print, for every hurricane, its *Name* and *Width* in a sentence format to the screen. If a second argument is passed to the function, it is a file identifier for an output file (which means that the file has already been opened), and the function will print in the same format to this file (and does not close it).

9) The built-in function **date** returns a string containing the day, month, and year. Write a function (using the **date** function) that will always return the current day. If the function call expects two output arguments, it will also return the month. If the function call expects three output arguments, it will also return the year.

whatdate.m

```
function [day, varargout] = whatdate
% Returns the current day and possibly also the
% current month and year
% Format of call: whatdate or whatdate()
% Returns the day; if 2 output arguments are expected,
% also the month; if 3 expected, also the year

d = date;

% always returns the day
[day, rest] = strtok(d, '-');

if nargout > 1
    % return the month also
    [month, rest] = strtok(rest, '-');
    varargout{1} = month;
```

```
end
if nargout == 3
    % return the year also
    varargout{2} = rest(2:end);
end
end
```

10) List some built-in functions to which you pass a variable number of input arguments (Note: this is not asking for **varargin**, which is a built-in cell array, or **nargin**.)

input, plot, randi, zeros, ones, etc, etc, etc; there are tons!

11) List some built-in functions that have a variable number of output arguments (or, at least one!).

size

12) Write a function that will receive a variable number of input arguments: the length and width of a rectangle, and possibly also the height of a box that has this rectangle as its base. The function should return the rectangle area if just the length and width are passed, or also the volume if the height is also passed.

boxcalcs.m

```
function [area, varargout] = boxcalcs(len, wid, varargin)
% Calculates a rectangle area and possibly also a
% height of a box and returns the area and possibly volume
% Format of call: boxcalcs(length, width) or
%   boxcalcs(length, width, height)
% Returns area of box and if height is passed, volume

% always return the area
area = len * wid;

n = nargin;

% if the height was passed, return the box volume
if nargin == 3
    varargout{1} = area * varargin{1};
end
end
```

13) Write a function to calculate the volume of a cone. The volume V is V = AH where A is the area of the circular base ($A = \pi r^2$ where r is the radius) and H is the height. Use a nested function to calculate A.

nestConeVol.m

```
function outvol = nestConeVol(rad, ht)
% Calculates the volume of a cone
% uses a nested function to calculate the area of
% the circular base of the cone
% Format of call: nestConeVol(radius, height)
% Returns the volume of the cone

outvol = conebase * ht;

function outbase = conebase
% calculates the area of the base
% Format of call: conebase or conebase()
% Returns the area of the circular base

outbase = pi*rad^2;
end % inner function

end % outer function
```

14) The two real roots of a quadratic equation $ax^2 + bx + c = 0$ (where a is nonzero) are given by

$$\frac{-b \pm \sqrt{D}}{2*a}$$

where the discriminant $D = b^2 - 4*a*c$. Write a function to calculate and return the roots of a quadratic equation. Pass the values of a, b, and c to the function. Use a nested function to calculate the discriminant.

quadeq.m

```
function [root1, root2] = quadeq(a,b,c)
% Calculates the roots of a quadratic equation
% ignores potential errors for simplicity
% Format of call: quadeq(a,b,c)
% Returns the two roots

d = discr;

root1 = (-b + sqrt(d))/(2*a);
root2 = (-b - sqrt(d))/(2*a);

function outd = discr
% calculates the discriminant
% Format of call: discr or discr()
% Returns the discriminant
```

```
outd = b^2 - 4*a*c;
end % inner function

end % outer function
```

15) The velocity of sound in air is 49.02 $\sqrt{T}$ feet per second where *T* is the air temperature in degrees Rankine. Write an anonymous function that will calculate this. One argument, the air temperature in degrees R, will be passed to the function and it will return the velocity of sound.

```
>> soundvel = @ (Rtemp) 49.02 * sqrt(Rtemp);
```

16) Create a set of anonymous functions to do length conversions and store them in a file named *lenconv.mat*. Call each a descriptive name, such as *cmtoinch* to convert from centimeters to inches.

```
>> clear
>> cmtoinch = @ (cm) cm/2.54;
>> inchtocm = @ (inch) inch * 2.54;
>> inchtoft = @ (inch) inch/12;
>> fttoinch = @ (ft) ft * 12;
>> save lenconv
>> clear
>> load lenconv
>> who
Your variables are:
cmtoinch  fttoinch  inchtocm  inchtoft
```

17) Write an anonymous function to convert from fluid ounces to milliliters. The conversion is one fluid ounce is equivalent to 29.57 milliliters.

```
floztoml = @(floz) floz * 29.57;
```

18) Why would you want to use an anonymous function?

They're simpler, don't have to use an M-file

19) Write an anonymous function to implement the following quadratic: $3x^2-2x+5$. Then, use **fplot** to plot the function in the range from -6 to 6.

```
>> quadfn = @ (x) 3*x^2 - 2*x + 5;
>> fplot(quadfn,[-6 6])
```

20) Write a function that will receive data in the form of x and y vectors, and a handle to a plot function and will produce the plot. For example, a call to the function would look like `wsfn(x,y,@bar)`.

wsfn.m

```
function wsfn(x,y,funhan)
% Plots funhan of x and y
% Format of call: wsfn(x,y,plot function handle)
% Does not return any values

funhan(x,y)
end
```

21) Write a function *plot2fnhand* that will receive two function handles as input arguments, and will display in two Figure Windows plots of these functions, with the function names in the titles. The function will create an x vector that ranges from 1 to n (where n is a random integer in the inclusive range from 4 to 10). For example, if the function is called as follows

>> plot2fnhand(@sqrt, @exp)

and the random integer is 5, the first Figure Window would display the **sqrt** function of x = 1:5, and the second Figure Window would display **exp(x)** for x = 1:5.

plot2fnhand.m

```
function plot2fnhand(funh1, funh2)
% Plots 2 function handles in 2 Figure Windows
% Format of call: plot2fnhand(fn hand1, fn hand2)
% Does not return any values

x = 1:randi([4, 10]);
figure(1)
y = funh1(x);
plot(x,y, 'ko')
title(func2str(funh1))
figure(2)
y = funh2(x);
plot(x,y,'ko')
title(func2str(funh2))
end
```

22) Use **feval** as an alternative way to accomplish the following function calls:

abs(-4)

size(zeros(4)) Use **feval** twice for this one!

```
>> feval(@abs, -4)

>> feval(@size, feval(@zeros, 4))
```

23) There is a built-in function function called **cellfun** that evaluates a function for every element of a cell array. Create a cell array, then call the **cellfun** function, passing the handle of the **length** function and the cell array to determine the length of every element in the cell array.

```
>> mycell = {'hello', 123, 'x', [4 5 33]};

>> cellfun(@length, mycell)
```

24) A recursive definition of a^n where a is an integer and n is a non-negative integer follows:

$a^n = 1$ if $n == 0$

$= a * a^{n-1}$ if $n > 0$

Write a recursive function called *mypower,* which receives a and n and returns the value of a^n by implementing the previous definition. Note: The program should NOT use ^ operator anywhere; this is to be done recursively instead! Test the function.

mypower.m

```
function res = mypower(a,n)
% recursively finds a^n
% Format of call: mypower(a,n)
% Returns a^n

if n == 0
    res = 1;
else
    res = a * mypower(a,n-1);
end
end
```

25) What does this function do:

```
function outvar = mystery(x,y)
if y == 1
   outvar = x;
else
   outvar = x + mystery(x,y-1);
end
```

Give one word to describe what this function does with its two arguments.

Multiplies!

The Fibonacci numbers is a sequence of numbers 0, 1, 1, 2, 3, 5, 8, 13, 21, 34... The sequence starts with 0 and 1. All other Fibonacci numbers are obtained by adding the previous two Fibonacci numbers. The higher up in the sequence that you go, the closer the fraction of one Fibonacci number divided by the previous is to the golden ratio. The Fibonacci numbers can be seen in an astonishing number of examples in nature, for example, the arrangement of petals on a sunflower.

26) The Fibonacci numbers is a sequence of numbers F_i:

0 1 1 2 3 5 8 13 21 34 …

where F_0 is 0, F_1 is 1, F_2 is 1, F_3 is 2, and so on. A recursive definition is:

$F_0 = 0$

$F_1 = 1$

$F_n = F_{n-2} + F_{n-1}$ if n > 1

Write a recursive function to implement this definition. The function will receive one integer argument n, and it will return one integer value that is the n^{th} Fibonacci number. Note that in this definition there is one general case but two base cases. Then, test the function by printing the first 20 Fibonacci numbers.

fib.m

```
function outval = fib(n)
% Recursively calculates the nth Fibonacci number
% Format of call: fib(n)
% Returns the nth Fibonacci number

if n == 0
    outval = 0;
elseif n == 1
    outval = 1;
else
    outval = fib(n-2) + fib(n-1);
end
end
```

```
>> for i = 1:20
   fprintf('%d\n', fib(i))
end
```

27) Use **fgets** to read strings from a file and recursively print them backwards.

printLinesRev.m

```
function printLinesRev
% Opens a file and call function to recursively
% print the lines in reverse order
% Format of call: printLinesRev or printLinesRev()
% Does not return any values

% Open file and check file open
fid = fopen('fileostrings.dat');
if fid == -1
    disp('File open not successful')
else
    fprtback(fid)
end
end

function fprtback(fid)
% Recursively print lines from file backwards
% Format of call: fprtback(fid)
% Does not return any values

aline = '';
if ~feof(fid)
    aline = fgets(fid);
    fprtback(fid)
end
disp(aline)
end
```

Chapter 11: Introduction to Object-Oriented Programming and Graphics

Exercises

1) Create a **double** variable. Use the functions **methods** and **properties** to see what are available for the class **double**.

```
>> number = 33.3;
>> methods(number)

Methods for class double:

abs             chebyshevU      erfinv          lambertw        rdivide
accumarray      chol            euler           ldivide         real
acos            cholupdate      exist           ldl             reallog
acosd           colon           exp             le              realpow
acosh           conj            expm1           legendreP
```

```
     etc.
>> properties(number)
No properties for class double.
```

2) Create a simple **plot** and store the handle in a variable. Use the three different methods (dot notation, **set**, and structure) to change the Color property.

```
>> x = -pi:0.01:2*pi;
>> y = sin(x);
>> ph = plot(x,y,'r*')
ph =
  Line with properties:

              Color: [1 0 0]
          LineStyle: 'none'
          LineWidth: 0.5000
             Marker: '*'
         MarkerSize: 6
    MarkerFaceColor: 'none'
              XData: [1x943 double]
              YData: [1x943 double]
              ZData: [1x0 double]

  Show all properties
>> set(ph,'Color','g')
>> ph.Color = [1 1 0];
>> fstruct = get(ph) % to see Color only
fstruct =
                Color: [1 1 0]
```

3) Create a **bar** chart and store the handle in a variable. Change the EdgeColor property to red.

```
>> y = [33 11 2 7 39];
>> hb = bar(y)
hb =
  Bar with properties:

    BarLayout: 'grouped'
     BarWidth: 0.8000
    FaceColor: 'flat'
    EdgeColor: [0 0 0]
    BaseValue: 0
        XData: [1 2 3 4 5]
        YData: [33 11 2 7 39]
```

```
  Show all properties
>> hb.EdgeColor = 'r'
hb =
  Bar with properties:

    BarLayout: 'grouped'
     BarWidth: 0.8000
    FaceColor: 'flat'
    EdgeColor: [1 0 0]
    BaseValue: 0
        XData: [1 2 3 4 5]
        YData: [33 11 2 7 39]

  Show all properties
```

4) Create a class *circleClass* that has a property for the radius of the circle and a constructor function. Make sure that there is a default value for the radius, either in the properties block or in the constructor. Instantiate an object of your class and use the **methods** and **properties** functions.

circleClass.m

```
classdef circleClass
   properties
       rad = 1;
   end

   methods
       %Constructor Function
       function obj = circleClass(varargin)
          if nargin ==1
             obj.rad = varargin{1};
          elseif nargin >= 2
               error('Incorrect number of input values')
          end
       end

   end


end
```

```
>> mycircle = circleClass
mycircle =
  circleClass with properties:

    rad: 1
```

```
>> yourcirc = circleClass(4.4)
yourcirc =
  circleClass with properties:

    rad: 4.4000
>> methods(yourcirc)

Methods for class circleClass:

circleClass

>> properties(mycircle)
Properties for class circleClass:
    rad
```

5) Add ordinary methods to *circleClass* to calculate the area and circumference of the circle.

circleClassii.m

```
classdef circleClassii
   properties
       rad = 1;
   end

   methods
       % Constructor
       function obj = circleClassii(varargin)
          if nargin >= 1
             obj.rad = varargin{1};
          end
       end

       % Ordinary methods

       function area = cirArea(obj)
           area = pi * (obj.rad) ^ 2;
       end

       function circum = cirCircum(obj)
           circum = 2 * pi * obj.rad;
       end
   end
end
```

```
>> newcircle = circleClassii(3.3)
newcircle =
  circleClassii with properties:
```

```
    rad: 3.3000
>> methods(newcircle)

Methods for class circleClassii:

cirArea        cirCircum      circleClassii

>> newcircle.cirArea
ans =
   34.2119
>> cirCircum(newcircle)
ans =
   20.7345
```

6) Create a class that will store the price of an item in a store, as well as the sales tax rate. Write an ordinary method to calculate the total price of the item, including the tax.

classTax.m

```
classdef classTax
    properties
        price
        rate = 6.25
    end

    methods

        function obj = classTax(varargin)
            if nargin == 0
                obj.price = 100;
            elseif nargin == 1
                obj.price = varargin{1};
            else
                obj.price = varargin{1};
                obj.rate = varargin{2};
            end
        end

        function tot = total(obj)
            tot = obj.price + obj.price * obj.rate/100;
        end
    end
end
```

```
>> amount = classTax
amount =
```

```
  classTax with properties:

    price: 100
     rate: 6.2500
>> purchase = classTax(9.99)
purchase =
  classTax with properties:

    price: 9.9900
     rate: 6.2500
>> bought = classTax(1000, 5)
bought =
  classTax with properties:

    price: 1000
     rate: 5
>> purchase.total
ans =
   10.6144
```

7) Create a class designed to store and view information on software packages for a particular software superstore. For every software package, the information needed includes the item number, the cost to the store, the price passed on to the customer, and a code indicating what kind of software package it is (e.g., ‘c’ for a compiler, ‘g’ for a game, etc.). Include a member function *profit* that calculates and prints the profit on a particular software product.

softwareClass.m

```
classdef softwareClass
    properties
        item_no = 123
        cost = 19.99
        price = 24.99
        code = 'x'
    end

    methods

        function obj = softwareClass(it, cost, p, code)
            % not handling varargin for simplicity
            if nargin == 4
                obj.item_no = it;
                obj.cost = cost;
                obj.price = p;
                obj.code = code;
            end
```

```
        end

        function prof = profit(obj)
            prof = obj.price - obj.cost;
            fprintf('The profit is $%.2f\n', prof)
        end
    end
end
```

```
>> xfin = softwareClass
xfin =
  softwareClass with properties:

    item_no: 123
       cost: 19.9900
      price: 24.9900
       code: 'x'
>> howmuch = xfin.profit;
The profit is $5.00
>> pack = softwareClass(111, 5, 50, 'g');
>> howmuch = pack.profit;
The profit is $45.00
>>
```

8) Create the *Rectangle* class from this chapter. Add a function to overload the **gt** (greater than) operator. Instantiate at least two objects and make sure that your function works.

Rectangle.m

```
classdef Rectangle

    properties
        len = 0;
        width = 0;
    end

    methods

        function obj = Rectangle(l, w)
            if nargin == 2
                obj.len = l;
                obj.width = w;
            end
        end

        function outarg = rectarea(obj)
```

```
            outarg = obj.len * obj.width;
        end

        function disp(obj)
            fprintf('The rectangle has length %.2f', obj.len)
            fprintf(' and width %.2f\n', obj.width)
        end

        function out = lt(obja, objb)
            out = rectarea(obja) < rectarea(objb);
        end

        function out = gt(obja, objb)
            out = rectarea(obja) > rectarea(objb);
        end
    end
end
```

```
>> rect1 = Rectangle(3, 5);
>> rect2 = Rectangle(2, 10);
>> rect1 > rect2
ans =
     0
>> gt(rect2, rect1)
ans =
     1
```

9) Create a class *MyCourse* that has properties for a course number, number of credits, and grade. Overload the **disp** function to display this information.

MyCourse.m

```
classdef MyCourse
    properties
        course_no = 'EK 127';
        credits = 4;
        grade = 'A';
    end

    methods

        function obj = MyCourse(cn, cr, gr)
            if nargin == 3
                obj.course_no = cn;
                obj.credits = cr;
                obj.grade = gr;
```

```
            end
        end

        function disp(obj)
          fprintf('The grade in the %d credit class %s was %s\n',...
                obj.credits, obj.course_no, obj.grade)
        end
    end
end
```

```
>> course1 = MyCourse
course1 =
The grade in the 4 credit class EK 127 was A
>> course2 = MyCourse('ME 333',4,'A-')
course2 =
The grade in the 4 credit class ME 333 was A-
```

10) Construct a class named *Money* that has 5 data members for dollars, quarters, dimes, nickels, and pennies. Include an ordinary function *equivtotal* that will calculate and return the equivalent total of the properties in an object (e.g., 5 dollars, 7 quarters, 3 dimes, 0 nickels and 6 pennies is equivalent to 7.11). Overload the **disp** function to display the properties.

Money.m

```
classdef Money
    properties
        dollars = 0;
        quarters = 0;
        dimes = 0;
        nickels = 0;
        pennies = 0;
    end

    methods

        function obj = Money(d, q, di, n, p)
            if nargin == 5
                obj.dollars = d;
                obj.quarters = q;
                obj.dimes = di;
                obj.nickels = n;
                obj.pennies = p;
            end
        end
```

```
        function total = equivtotal(obj)
            total = obj.dollars + obj.quarters*.25 + ...
                obj.dimes*.1 + obj.nickels*.05 + obj.pennies*.01;
        end

        function disp(obj)
            fprintf('The total dollar amount is $%.2f\n', ...
                equivtotal(obj))
        end
    end
end
```

```
>> amt = Money
amt =
The total dollar amount is $0.00
>> amount = Money(2, 1, 3, 4, 3)
amount =
The total dollar amount is $2.78
>>
```

11) Write a program that creates a class for complex numbers. A complex number is a number of the form a + bi, where a is the real part, b is the imaginary part, and i = $\sqrt{-1}$. The class *Complex* should have properties for the real and imaginary parts. Overload the **disp** function to print a complex number.

Complex.m

```
classdef Complex
    properties
        realpart = 0;
        imagpart = 0;
    end

    methods

        function obj = Complex(rp, ip)
            if nargin == 2
                obj.realpart = rp;
                obj.imagpart = ip;
            end
        end

        function disp(obj)
            fprintf('%.1f + %.1fi\n', obj.realpart, obj.imagpart)
        end
    end
```

```
end
```

```
>> compnum = Complex(2,4)
compnum =
2.0 + 4.0i
>>
```

12) Create a base class *Square* and then a derived class *Cube*, similar to the Rectangle/Box example from the chapter. Include a function to calculate the area of a square and volume of a cube.

Square.m

```
classdef Square

    properties
        side = 1;
    end

    methods

        function obj = Square(s)
            if nargin == 1
                obj.side = s;
            end
        end

        function outarg = area(obj)
            outarg = obj.side ^ 2;
        end

        function disp(obj)
            fprintf('The square has side %.2f\n', obj.side)
        end

    end
end
```

Cube.m

```
classdef Cube < Square

    % properties are not needed

    methods
        function obj = Cube(s)
            if nargin == 0
                s = 1;
```

```
            end
            obj@Square(s)
        end

        function out = volume(obj)
            out = obj.side ^ 3;
        end

        function disp(obj)
            fprintf('The cube has volume %.1f\n', volume(obj))
        end
    end
end
```

```
>> mys = Square(3)
mys =
The square has side 3.00
>> mys.area
ans =
     9
>> myc = Cube(4)
myc =
The cube has volume 64.0
```

13) Create a base class named *Point* that has properties for x and y coordinates. From this class derive a class named *Circle* having an additional property named *radius*. For this derived class the x and y data members represent the center coordinates of a circle. The function members of the base class should consist of a constructor, an area function that returns 0, and a distance function that returns the distance between two points ($\text{sqrt}((x2-x1)^2 + (y2-y1)^2)$). The derived class should have a constructor and an override function named area that returns the area of a circle. Write a script that has 2 objects of each class and calls all of the member functions.

Point.m

```
classdef Point
    properties
        x = 0;
        y = 0;
    end

    methods

        function obj = Point(xc, yc)
            if nargin == 2
```

```
                obj.x = xc;
                obj.y = yc;
            end
        end

        function out = area(obj)
            out = 0;
        end

        function out = distance(obja, objb)
            out = sqrt((objb.x - obja.x) ^2 + ...
                (objb.y - obja.y)^2);
        end
    end
end
```

Circle.m

```
classdef Circle < Point
    properties
        radius = 0;
    end

    methods

        function obj = Circle(xc, yc, r)
            if nargin < 3
                xc = 0;
                yc = 0;
                r = 0;
            end
            obj@Point(xc,yc)
            obj.radius = r;
        end

        function out = area(obj)
            out = pi * obj.radius ^ 2;
        end
    end
end
```

Ch11Ex13.m

```
pta = Point(2.5,5)
ptb = Point(3, 6)
distance(pta,ptb)
pta.distance(ptb)
pta.area
onecirc = Circle(2,5,6);
```

```
onecirc.area
```

14) Take any value class (e.g., *MyCourse* or *Square*) and make it into a handle class. What are the differences?

SquareH.m

```
classdef SquareH < handle

    properties
        side = 1;
    end

    methods

        function obj = SquareH(s)
            if nargin == 1
                obj.side = s;
            end
        end

        function outarg = area(obj)
            outarg = obj.side ^ 2;
        end

        function disp(obj)
            fprintf('The square has side %.2f\n', obj.side)
        end

    end
end
```

```
>> vsq1 = Square(3)
vsq1 =
The square has side 3.00
>> vsq2 = vsq1
vsq2 =
The square has side 3.00
>> vsq1.side = 5
vsq1 =
The square has side 5.00
>> vsq2
vsq2 =
The square has side 3.00
>>
>> hsq1 = SquareH(3)
```

```
hsq1 =
The square has side 3.00
>> hsq2 = hsq1
hsq2 =
The square has side 3.00
>> hsq1.side = 5
hsq1 =
The square has side 5.00
>> hsq2
hsq2 =
The square has side 5.00
```

15) Create a class that stores information on a company's employees. The class will have properties to store the employee's name, a 10-digit ID, their department and a rating from 0 to 5. Overwrite the *set.propertyname* function to check that each property is the correct class and that:

- The employee ID has 10 digits
- The department is one of the following codes: HR (Human Resources), IT (Information Technology), MK (Marketing), AC (Accounting), or RD (research and Development)
- The rating is a number from 0 to 5.

The rating should not be accessible to anyone without a password. Overwrite the *set.rating* and *get.rating* functions to prompt the user for a password. Then, write a function that returns the rating.

```
classdef employeeData
   properties (SetAccess = protected)
       name
       id
       depart
   end

   properties (Access = private)
       rating
   end

   methods
       function obj = employeeData(varargin)
           if nargin == 1
               obj.name = varargin{1};
           elseif nargin == 2
               obj.name = varargin{1};
               obj.id = varargin{2};
           elseif nargin ==3
               obj.name = varargin{1};
```

```
        obj.id = varargin{2};
        obj.depart = varargin{3};
    elseif nargin ==4
        obj.name = varargin{1};
        obj.id = varargin{2};
        obj.depart = varargin{3};
        obj.rating = varargin{4};
    else
        error('Invalid number of inputs')
    end

end

function obj = set.name(obj,val)
    if isa(val,'char')
        obj.name = val;
    else
        error('Not a valid name')
    end
end

function obj = set.id(obj,val)
    if isa(val,'double')&&numel(num2str(val)) == 10
            obj.id = num2str(val);
    else
        error('Not a valid ID number')
    end
end

function obj = set.depart(obj, val)
    if strcmp(val,'HR')
        obj.depart = val;
    elseif strcmp(val,'IT')
        obj.depart = val;
    elseif strcmp(val,'RD')
        obj.depart = val;
    elseif strcmp(val,'AC')
        obj.depart = val;
    elseif strcmp(val,'MK')
        obj.depart = val;
    else
        error('Not a valid department')
    end
end

function obj = set.rating(obj,val)
    code = input('Password Required:');
```

```
            if code == 1234
                if isa(val,'double') && val>=0 && val<=5
                obj.rating = val;
                else
                    error('Invalid rating input')
                end
            else
                error('Invalid Password')
            end
        end

        function val = get.rating(obj)
            code = input('Password Required:');
            if code == 1234
                val = obj.rating;
            else
                error('Invalid Password')
            end
        end

        function rat = seeRating(obj)
            rat = obj.rating;
        end

    end
end
```

16) Create a handle class that logs the times a company's employees arrive and leave at work. The class must have the following characteristics:

- As the employer, you do not want your employees to access the information stored.
- The class will store date, hour, minute, second and total time as properties.
- The constructor function will input the data from the *clock* function, which returns a vector with format [year month day hour minute second].
- Each time an employee arrives or leaves, they must call a method LogTime that will store the new times with the old times.

Include a method calcPay that calculates the money owed if it is assumed that the employees are paid $15/hour. In order to do this, call a separate method that calculates the time elapsed between the last two time entries. Use the function **etime**. This method should only be available to call by *calcPay*, and the existence of calcPay should only be known to the coder.

timeLog.m

```
classdef timeLog < handle
    properties (SetAccess = protected)
        date
        hour
        min
        sec
        time = 0;
    end

    methods
        function obj = timeLog
            t = clock;
            obj.date = t(1:3);
            obj.hour =  t(4);
            obj.min = t(5);
            obj.sec = t(6);
        end

        function LogTime(obj)
            t = clock;
            obj.date = [obj.date; t(1:3)];
            obj.hour = [obj.hour; t(4)];
            obj.min = [obj.min; t(5)];
            obj.sec = [obj.sec; t(6)];
        end
    end
    methods (Access= protected)
        function  t = calcTime(obj)
            timevec1 = [obj.date(end,:), obj.hour(end),...
                obj.min(end),obj.sec(end)];
            timevec2=[obj.date(end-1,:),obj.hour(end-1),...
                obj.min(end-1),obj.sec(end-1)];
            t = etime(timevec1,timevec2);
        end
    end
    methods (Hidden)
        function mon = calcPay(obj)
            %Paid 15$/hour. Calculate $/second
            obj.time = obj.time + calcTime(obj);
            rate = 15/3600;
            mon = obj.time*rate;
        end
    end
end
```

17) You head a team developing a small satellite in competition for a NASA contract. Your design calls for a central satellite that will deploy sensor nodes. These nodes must remain within 30 km of the satellite to allow for data transmission. If they pass out of range, they will use an impulse of thrust propulsion to move back towards the satellite. Make a *Satellite* class with the following properties:

- *location*: An [X Y Z] vector of coordinates, with the satellite as the origin.
- *magnetData*: A vector storing magnetic readings.
- *nodeAlerts*: An empty string to begin with, stores alerts when nodes go out of range.

Satellite also has the following methods:

- *Satellite*: The constructor, which sets location to [0 0 0] and magnetData to 0.
- *retrieveData*: Takes data from a node, extends the magnetData vector.

Then, make the *sensorNode* class as a subclass of Satellite. It will have the following properties:

- *distance*: The magnitude of the distance from the satellite. Presume that a node's location comes from on-board, real-time updating GPS (i.e., do not worry about updating node.location).
- *fuel*: Sensor nodes begin with 100 kg of fuel.

sensorNode also has the following methods:

- *sensorNode*: The constructor.
- *useThrust*: Assume this propels node towards satellite. Each usage consumes 2 kg of fuel. If the fuel is below 5 kg, send an alert message to the satellite.
- *checkDistance*: Check the magnitude of the distance between
- useMagnetometer: Write this as a stub. Have the "magnetometer reading" be a randomized number in the range 0 to 100.
- *sendAlert*: set the "nodeAlerts" Satellite property to the string 'Low fuel'.

First, treat both classes as value classes. Then, adjust your code so that both are handle classes. Which code is simpler?

Solution 1: Value class

```
classdef Satellite

    properties
        location;
        magnetData;
        nodeAlerts;
    end

    methods
```

```
        %Constructor
        function obj = Satellite(varargin)
            if nargin == 2
                location = varargin{1};
                magnetData = varargin{2};
            else
                location = [0 0 0];
                 %the satellite is the origin point, by default
                magnetData = 0; %initialize to 0
            end
        end
        %takes data from node, stores it aboard the satellite
        function obj = retrieveData(sat,node)
            node = node.useMagnetometer(node);
            obj.magnetData = sat.magnetData + node.magnetData;
        end
    end
end


classdef sensorNode < Satellite

    properties
        distance = 0;
        fuel = 100; %Nodes start with 100kg of fuel.

    end

    methods

        function obj = sensorNode(varargin)
        %Constructor
            if nargin == 2
                distance = varagin{1};
                fuel = varagin{2};
            elseif nargin == 4
                obj@Satellite(varargin{3},varagin{4});
                distance = varagin{1};
                fuel = varagin{2};
            end
        end

        function [obj1, obj2] = useThrust(sat, node)
        %Use a brief impulse of thrust to move node back towards
        % satellite. Assume thruster
        % controls correctly determine how to move closer.
        %Every time thrust is used, 2 kg of fuel is consumed.
            node.fuel = node.fuel - 2;
            if node.fuel <= 10
                  sat = node.sendAlert(sat);
            end
            obj1 = node;
            obj2 = sat;
        end

        function obj = checkDistance(sat,node)
```

```
        %Check distnace between nodes and satellite.
            distXYZ = sat.location - node.location;
            node.distance = sqrt(distXYZ(1)^2 + distXYZ(2)^2 + ...
                    distXYZ(3)^2);
            if node.distance > 30
                  [node,sat] = node.useThrust(sat,node);
            end
            obj = node;
        end

        function obj = useMagnetometer(node)
        %Gather data from sensor hardware
        % the +10 is a stubbed value, substituting what the
        % sensor will read from its magnetometer.
            obj.magnetData = node.magnetData + 10;
        end

        function obj = sendAlert(sat)
        %Alerts the satellite that the node is out of range.
            sat.nodeAlerts = sprintf('Node out of fuel.');
            obj = sat;
        end

    end
```

Solution 2: Handle class

```
classdef SatelliteHx < handle

    properties
        location;
        magnetData;
        nodeAlerts;
    end

    methods

       %Constructor
       function obj = SatelliteHx(varargin)
           if nargin == 2
               location = varargin{1};
               magnetData = varargin{2};
           else
               location = [0 0 0];
                %the satellite is the origin point, by default
               magnetData = 0;
           end
       end

       function retrieveData(sat,node)
       %takes data from node, stores it aboard the satellite
           node.useMagnetometer(node);
           sat.magnetData = sat.magnetData + node.magnetData;
       end
```

```
    end
end


classdef sensorNode < Satellite

    properties
        distance = 0;
        fuel = 100; %Nodes start with 100kg of fuel.
    end
    methods

        function obj = sensorNode(varargin)
        %Constructor
            if nargin == 2
                distance = varagin{1};
                fuel = varagin{2};
            elseif nargin == 4
                obj@Satellite(varargin{3},varagin{4});
                distance = varagin{1};
                fuel = varagin{2};
            end
        end

        function useThrust(sat, node)
            node.fuel = node.fuel - 2;
            %Every time thrust is used, 2 kg of fuel is consumed.
            if node.fuel <= 10
                  sat = node.sendAlert(sat);
            end
        end

        function checkDistance(sat,node)
        %Check distnace between nodes and satellite.
            distXYZ = sat.location - node.location;
            node.distance = sqrt(distXYZ(1)^2 + distXYZ(2)^2 + ...
                 distXYZ(3)^2);
            if node.distance > 30
                  node.useThrust(sat,node);
            end
        end

        function useMagnetometer(node)
        %Gather data from sensor hardware
        %the +10 is a stubbed value, substituting what the
        %sensor will read from its magnetometer.
            node.magnetData = node.magnetData + 10;
        end
        %Gather
        function sendAlert(sat)
            sat.nodeAlerts = sprintf('Node out of fuel.');
        end

    end
```

Chapter 12: Advanced Plotting Techniques

Exercises

1) Create a data file that containing 10 numbers. Write a script that will load the vector from the file, and use **subplot** to do an **area** plot and a **stem** plot with these data in the same Figure Window (Note: a loop is not needed). Prompt the user for a title for each plot.

Ch12Ex1.m

```
% Subplot of area and stem plots
load data1.dat
subplot(1,2,1)
area(data1)
title1 = input('Enter a title of the area plot: ','s');
title(title1)
subplot(1,2,2)
stem(data1)
title2 = input('Enter a title of the stem plot: ','s');
title(title2)
```

2) Write a script that will read x and y data points from a file, and will create an **area** plot with those points. The format of every line in the file is the letter 'x', a space, the x value, space, the letter 'y', space, and the y value. You must assume that the data file is in exactly that format, but you may not assume that the number of lines in the file is known. The number of points will be in the plot title. The script loops until the end of file is reached, using **fgetl** to read each line as a string. For example, *if* the file contains the following lines,

```
x 0 y 1
x 1.3 y 2.2
x 2.2 y 6
x 3.4 y 7.4
```

when running the script, the result will be as shown in the Figure.

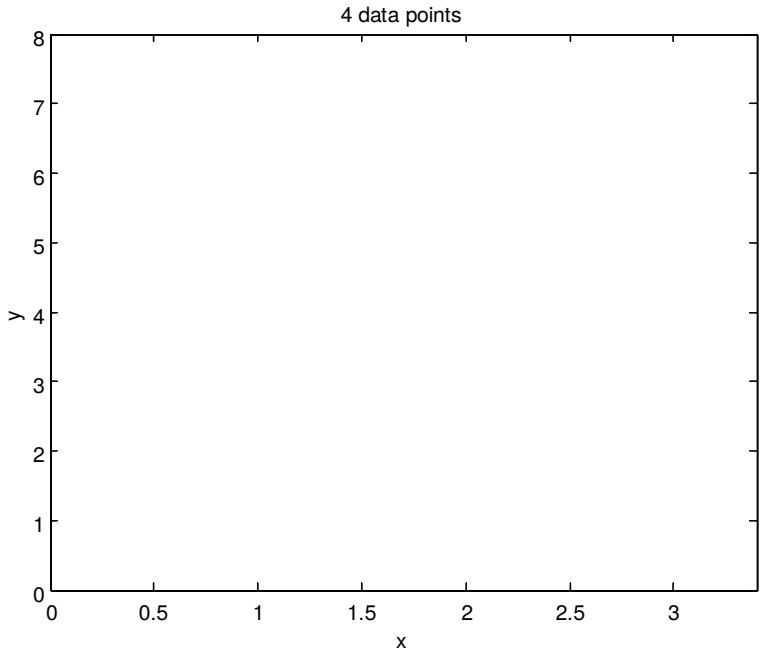


Figure Area plot produced from x, y data read as strings from a file

Ch12Ex2.m

```
% Read x and y coordinates of points from a data file
% and plot them using an area plot

x = [];
y = [];

fid = fopen('xandypts.dat');
if fid == -1
    disp('File open not successful')
else
  while ~feof(fid)
    aline = fgetl(fid);
    [letter, rest] = strtok(aline);
    [xval, rest] = strtok(rest);
    [letter, rest] = strtok(rest);
    yval = strtok(rest);
    x = [x str2num(xval)];
    y = [y str2num(yval)];
  end

  area(x,y)
  title(sprintf('%d data points',length(x)))

  fclose(fid);
end
```

3) Do a quick survey of your friends to find out who prefers cheese pizza, pepperoni, or mushroom (no other possibilities; everyone must pick one of those three choices). Draw a pie chart to show the percentage favoring each. Label the pieces of this pizza pie chart!

Ch12Ex3.m

```
% Pie chart showing how many people prefer
%  different types of pizzas

cheese = 7;
pep = 8;
mushroom = 11;
pizzas = [cheese pep mushroom];
pie(pizzas,{'Cheese','Pepperoni','Mushroom'})
title('Favorite pizzas')
```

4) The number of faculty members in each department at a certain College of Engineering is:

```
ME  22
BM  45
CE  23
EE  33
```

Experiment with at least 3 different plot types to graphically depict this information. Make sure that you have appropriate titles, labels, and legends on your plots. Which type(s) work best, and why?

Ch12Ex4.m

```
%Graphically depict faculty sizes using different plots

ME = 22;
BM = 45;
CE = 23;
EE = 33;
data = [ME BM CE EE];
labels = {'ME','BM','EE','CE'};

% plot is not very good method
plot(data)
set(gca,'XTick',1:4,'XTickLabel',labels)
title('College Faculty')

% pie chart allows comparison but if two numbers are
% close, virtually impossible to see which is larger
figure
pie(data,labels);
```

```
title('College Faculty')

% bar chart also easy to compare and can easily
% compare any two with each other
figure
bar(data)
set(gca,'XTickLabel',labels)
title('College Faculty')
```

5) Experiment with the **comet** function: try the example given when **help comet** is entered and then animate your own function using **comet**.

```
>> x = linspace(0,16*pi,1000);
>> y = exp(-x/10).*cos(x);
>> comet(x,y)
```

6) Experiment with the **comet3** function: try the example given when **help comet3** is entered and then animate your own function using **comet3**.

```
>> theta = linspace(0,12*pi,1000);
>> x = exp(-theta/(4*pi)).*cos(theta);
>> y = exp(-theta/(4*pi)).*sin(theta);
>> z = theta/(4*pi);
>> comet3(x,y,z)
```

7) Experiment with the **scatter** and **scatter3** functions.

Ch12Ex7.m

```
% Investigates the scatter and scatter3 plots
% using random sizes and colors for random points

clf
%Number of points
n = 50;
%Data
x = rand([1 n])*10;
y = rand([1 n])*10;
%Size and color
s = randi([50,1000],1,n);
c = randi([1,64],1,n);

figure(1)
scatter(x,y,s,c,'filled')
title('Random sizes and colors')
```

```
%Number of points
n = 50;
%Data
x = rand([1 n])*10;
y = rand([1 n])*10;
z = rand([1 n])*10;

%Size and color
s = randi([50,1000],1,n);
c = randi([1,64],1,n);

figure(2)
scatter3(x,y,z,s,c,'filled')
title('Random sizes and colors')
```

8) Use the **cylinder** function to create *x*, *y*, and *z* matrices and pass them to the **surf** function to get a surface plot. Experiment with different arguments to **cylinder**.

```
>> [x y z] = cylinder(2,30);
>> surf(x,y,z)
```

9) Experiment with **contour** plots.

```
>> [x y] = meshgrid(1:2, 1:3);
>> z = exp(x) .* sin(y);
>> contour(x,y,z)
```

10) The electricity generated by wind turbines annually in kilowatt-hours per year is given in a file. The amount of electricity is determined by, among other factors, the diameter of the turbine blade (in feet) and the wind velocity in mph. The file stores on each line the blade diameter, wind velocity, and the approximate electricity generated for the year. For example,

```
5 5 406
5 10 3250
5 15 10970
5 20 26000
10 5 1625
10 10 13000
10 15 43875
10 20 104005
```

Create a file in this format, and determine how to graphically display this data.

Ch12Ex10.m

```
% Read wind turbine data from a file and depict
% graphically using a 3D stem plot

load turbine.dat

stem3(turbine(:,1),turbine(:,2),turbine(:,3));
xlabel('Blade Diameter (ft)')
ylabel('Wind Velocity (mph)')
zlabel('Electricity Generated (kW-hr/yr)')
title('Wind turbine data')
```

11) Create an x vector, and then two different vectors (y and z) based on x. Plot them with a legend. Use **help legend** to find out how to position the legend itself on the graph, and experiment with different locations.

Ch12Ex11.m

```
% Create vectors, plot, and experiment with legend location

y = x.^2;
z = sqrt(x);
hold on
plot(x,y,'r-')
plot(x,z,'g-')
legend('x^2','sqrt(x)','Location','NorthWest')
```

12) Create an x vector that has 30 linearly spaced points in the range from -2 π to 2 π, and then y as **sin(x)**. Do a **stem** plot of these points, and store the handle in a variable. Use **get** to see the properties of the stem plot, and then **set** to change the face color of the marker. Also do this using the dot operator.

Ch12Ex12.m

```
% Display a stem plot and modify the face color
%  of the marker to red

x = linspace(-2*pi,2*pi,30);
y = sin(x);
hdl = stem(x,y);
xlabel('x')
ylabel('sin(x)')
title('Stem plot of sin')
get(hdl);
set(hdl,'MarkerFaceColor','r')
```

13) When an object with an initial temperature T is placed in a substance that has a temperature S, according to Newton's law of cooling in t minutes it will reach a temperature T_t using the formula $T_t = S + (T - S)\ e^{(-kt)}$ where k is a constant value that depends on properties of the object. For an initial temperature of 100 and k = 0.6, graphically display the resulting temperatures from 1 to 10 minutes for two different surrounding temperatures: 50 and 20. Use the **plot** function to plot two different lines for these surrounding temperatures, and store the handle in a variable. Note that two function handles are actually returned and stored in a vector. Change the line width of one of the lines.

Ch12Ex13.m

```
% Plot the cooling temperatures of an object placed in
% two different surrounding temperatures

time = linspace(1,10,100);
T1 = 50 + (100 - 50)*exp(-0.6*time);
T2 = 20 + (100 - 20)*exp(-0.6*time);
hdl = plot(time,T1,'b-',time,T2,'r-');
xlabel('Time')
ylabel('Temperature')
legend('50 degrees', '20 degrees')
set(hdl(1),'LineWidth',3)
```

14) Write a script that will draw the line y=x between x=2 and x=5, with a random line width between 1 and 10.

Ch12Ex14.m

```
% Plot line y = x with a random thickness

x = [2 5];
y = x;
hdl = plot(x,y);
title('Line with random thickness')
set(hdl,'LineWidth',randi([1 10]))
```

15) Write a script that will plot the data points from y and z data vectors, and store the handles of the two plots in variables *yhand* and *zhand*. Set the line widths to 3 and 4 respectively. Set the colors and markers to random values (create strings containing possible values and pick a random index).

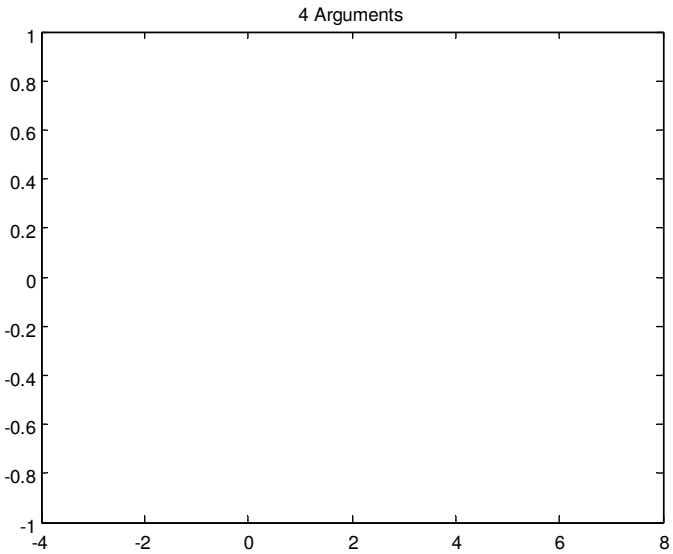
Ch12Ex15.m

```
y = [33 22 17 32 11];
z = [3 7 2 9 4 6 2 3];
```

```
figure(1)
yhand = plot(y);
figure(2)
zhand = plot(z);
somecolors = 'bgrcmyk';
somemarkers = '.ox+*sd';
set(yhand,'LineWidth',3, ...
      'Color',somecolors(randi(length(somecolors))))
set(zhand,'LineWidth',4,...
      'Color',somecolors(randi(length(somecolors))))
set(yhand, 'Marker',...
          somemarkers(randi(length(somemarkers))))
set(zhand, 'Marker', ...
          somemarkers(randi(length(somemarkers))))
```

16) Write a function *plotexvar* that will plot data points represented by x and y vectors which are passed as input arguments. If a third argument is passed, it is a line width for the plot, and if a fourth argument is also passed, it is a color. The plot title will include the total number of arguments passed to the function. Here is an example of calling the function and the resulting plot:

```
>> x=-pi:pi/50:2*pi;
>> y = sin(x);
>> plotexvar(x,y,12,'r')
```



plotexvar.m

```
function plotexvar(x,y,varargin)
n = nargin;
```

```
han = plot(x,y);
title(sprintf('%d Arguments',n))
if n==3
    set(han,'LineWidth',varargin{1})
elseif n==4
    set(han,'LineWidth',varargin{1},...
        'Color',varargin{2})
end
```

17) A file *houseafford.dat* stores on its three lines years, median incomes and median home prices for a city. The dollar amounts are in thousands. For example, it might look like this:

```
2004 2005 2006 2007 2008 2009 2010 2011
72 74 74 77 80 83 89 93
250 270 300 310 350 390 410 380
```

Create a file in this format, and then **load** the information into a matrix. Create a horizontal stacked bar chart to display the information, with an appropriate title. Note: use the 'XData' property to put the years on the axis as shown in Figure 11.35.

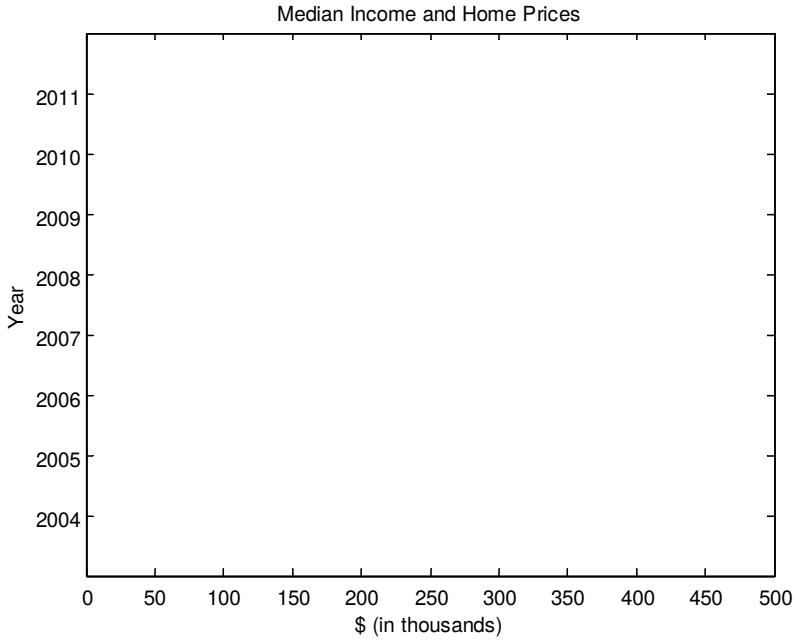


Figure 12.33 Horizontal stacked bar chart of median incomes and home prices

Ch12Ex17.m

```
% Read in median incomes and house prices from a file
% and plot using a stacked bar

load houseafford.dat
```

```
h = barh(houseafford(2:3,:)','stacked');
xlabel('$')
ylabel('Year')
set(h,'Xdata',houseafford(1,:))
title('Median Income and Home Prices')
```

18) Write a function that will plot **cos(x)** for x values ranging from –pi to pi in steps of 0.1, using black *'s. It will do this three times across in one Figure Window, with varying line widths (Note: even if individual points are plotted rather than a solid line, the line width property will change the size of these points.). If no arguments are passed to the function, the line widths will be 1, 2, and 3. If, on the other hand, an argument is passed to the function, it is a multiplier for these values (e.g., if 3 is passed, the line widths will be 3, 6, and 9). The line widths will be printed in the titles on the plots.

cosLineWidths.m

```
function cosLineWidths(varargin)
% Plots cos(x) 3 times using subplot with
% varying line widths
% Format of call: cosLineWidths or cosLineWidths()
%   or cosLineWidths(multiplier)

x = -pi: 0.1: pi;
y = cos(x);

multiplier = 1;
if nargin == 1
    multiplier = varargin{1};
end

for i = 1:3
    subplot(1,3,i)
    plot(x,y,'k*','LineWidth', i*multiplier)
    sptitle = sprintf('Line Width %d', i*multiplier);
    title(sptitle)
    xlabel('x')
    ylabel('cos(x)')
end
end
```

19) Create a graph, and then use the text function to put some text on it, including some \specchar commands to increase the font size and to print some Greek letters and symbols.

Ch12Ex19.m

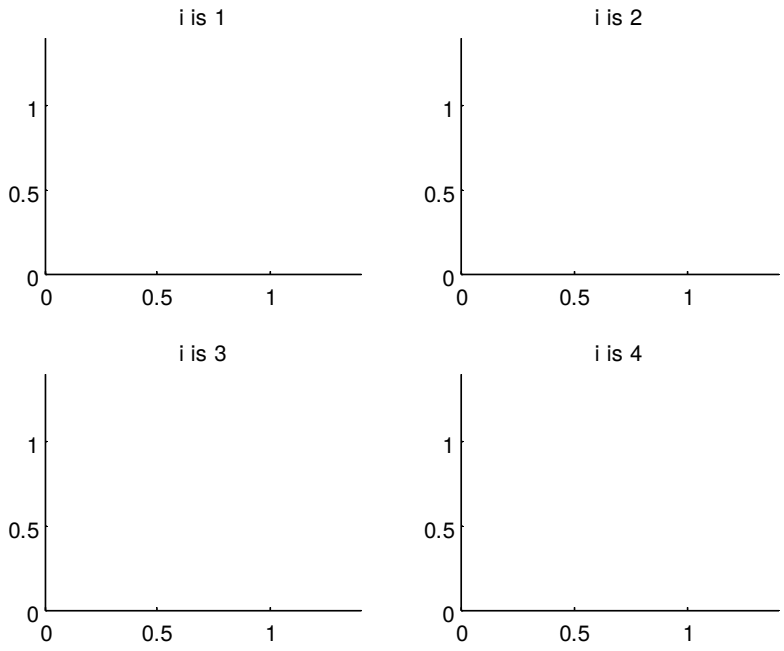
```
% Experiment with the text function and special
%  characters

x = -2:.1:5;
y  = cos(x);
plot(x,y)
text(0,-.5,'Random color and font',...
'FontSize', 20, 'Color', [.3 .5 .7])
text(0,0.2, 'Symbols: \epsilon \heartsuit')
```

20) Create a **rectangle** object, and use the **axis** function to change the axes so that you can see the rectangle easily. Change the Position, Curvature, EdgeColor, LineStyle, and LineWidth. Experiment with different values for the Curvature.

Ch12Ex20.m

```
% Experiment with a rectangle object and properties

rhand = rectangle;
axis([-1 2 -1 2])
set(rhand, 'Position', [-0.5, -0.5, 1, 1], ...
    'Curvature', [0.3, 0.4], 'EdgeColor', 'blue', ...
    'LineStyle', ':', 'LineWidth', 4)
title('Rectangle object')
```

21) Write a script that will display rectangles with varying curvatures and line widths, as shown in the Figure. The script will, in a loop, create a 2 by 2 subplot showing rectangles. In all, both the x and y axes will go from 0 to 1.4. Also, in all, the lower left corner of the rectangle will be at (0.2, 0.2), and the length and width will both be 1. The line width, i, is displayed in the title of each plot. The curvature will be [0.2, 0.2] in the first plot, then [0.4, 0.4], [0.6,0.6], and finally [0.8,0.8].



Figure

Ch12Ex21.m

```
% Display a 2 by 2 subplot of rectangles with
% varying curvatures and line widths

for i = 1:4
    subplot(2,2,i)
    rectangle('Position',[0.2 0.2 1 1], ...
        'LineWidth', i, ...
        'Curvature', [0.2*i 0.2*i])
    axis([0 1.4 0 1.4])
    title(sprintf('Line Width is %d', i))
end
```

22) Write a script that will start with a rounded rectangle. Change both the x and y axes from the default to go from 0 to 3. In a **for** loop, change the position vector by adding 0.1 to all elements 10 times (this will change the location and size of the rectangle each time). Create a movie consisting of the resulting rectangles. The final result should look like the plot shown in the Figure.

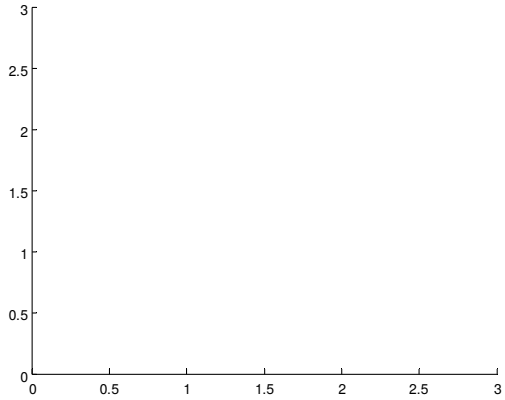


Figure Curved rectangles produced in a loop

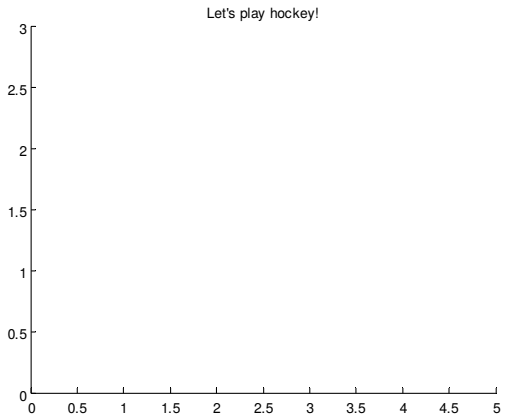
Ch12Ex22.m

```
% Displays a series of rounded rectangles

posvec = [0.2, 0.2, 0.5, 0.8];
rh = rectangle('Position', posvec,...
    'Curvature',[0.5, 0.5]);
axis([0 3 0 3])
set(rh,'Linewidth',3,'LineStyle',':')
for i = 1:10
    posvec = posvec + 0.1;
    rh = rectangle('Position', posvec,...
       'Curvature',[0.5, 0.5]);
    axis([0 3 0 3])
    set(rh,'Linewidth',3,'LineStyle',':')
end
```

23) A hockey rink looks like a rectangle with curvature. Draw a hockey rink, as in the Figure.

Ch12Ex23.m

```
% Draw a hockey rink

rectangle('Position', [0.5 0.5 4 2], ...
    'Curvature', [0.6 0.6])
axis([0 5 0 3])
line([2.5 2.5], [0.5 2.5], 'Color', 'r', ...
    'LineWidth', 4)
title('Let''s play hockey!')
```



24) Write a script that will create a two-dimensional **patch** object with just three vertices and one face connecting them. The x and y coordinates of the three vertices will be random real numbers in the range from 0 to 1. The lines used for the edges should be black with a width of 3, and the face should be grey. The axes (both x and y) should go from 0 to 1. For example, depending on what the random numbers are, the Figure Window might look like the Figure.

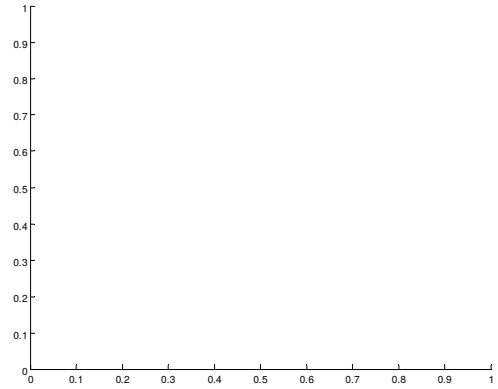


Figure Patch object with black edge

Ch12Ex24.m

```
% Create a 2D patch object with random vertices
% and just one face

polyhedron.vertices = [...
rand rand rand
rand rand rand
rand rand rand];

polyhedron.faces = [1 2 3];

pobj = patch(polyhedron, ...
   'FaceColor',[0.8, 0.8, 0.8],...
```

```
   'EdgeColor','black', 'LineWidth', 3);
axis([0 1 0 1])
```

25) Using the **patch** function, create a black box with unit dimensions (so, there will be eight vertices and six faces). Set the edge color to white so that when you rotate the figure, you can see the edges.

Ch12Ex25.m

```
% Draws a black box with unit dimensions

% set up the 8 vertices
polyhedron.vertices = [...
0 0 0
1 0 0
1 1 0
0 1 0
0 0 1
1 0 1
1 1 1
0 1 1];

% connect the 6 faces
polyhedron.faces = [...
    1 2 3 4
    5 6 7 8
    1 2 6 5
    1 4 8 5
    2 3 7 6
    3 4 8 7];

pobj = patch(polyhedron, ...
'FaceColor',[0 0 0],...
'EdgeColor','white');
```

26) Write a function *plotline* that will receive x and y vectors of data points, and will use the **line** primitive to display a line using these points. If only the x and y vectors are passed to the function, it will use a line width of 5; otherwise, if a third argument is passed, it is the line width.

plotline.m

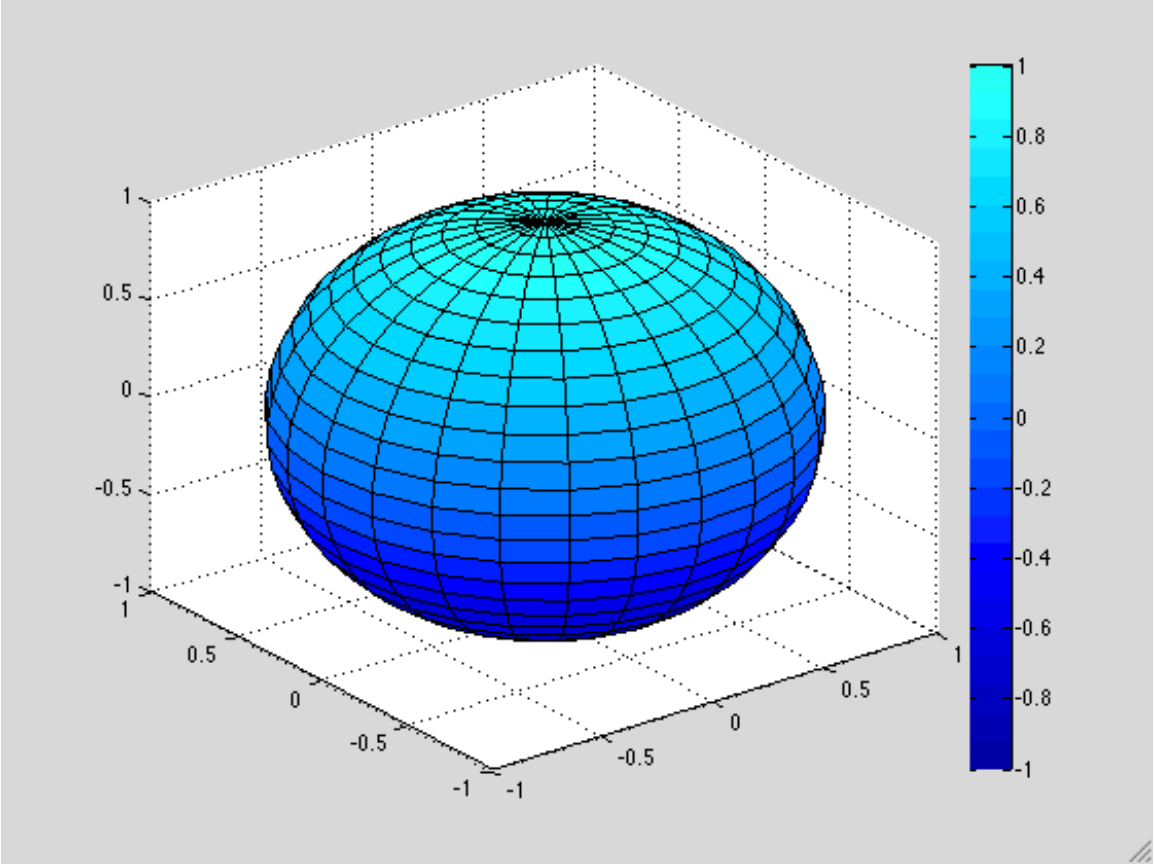
```
function plotline(x,y,varargin)
lw = 5;
if nargin == 3
    lw = varargin{1};
end
```

```
line(x,y,'LineWidth',lw)
end
```

Chapter 13: Sights and Sounds

Exercises

1) Create a custom colormap for a sphere that consists of the first 25 colors in the default colormap **jet**. Display **sphere(25)** with a colorbar.




```
>> cmap = colormap;
>> cmap = cmap(1:25,:);
>> colormap(cmap)
>> [x, y, z] = sphere(25);
>> surf(x,y,z)
>> colorbar
```

2) Write a script that will create the image seen in the Figure below using a colormap.

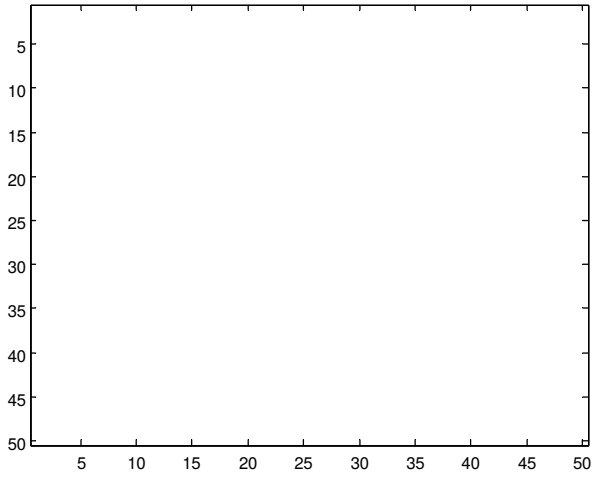


Figure Image displaying four colors using a custom colormap

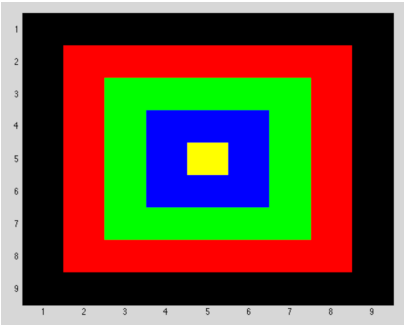
Ch13Ex2.m

```
% Creates a four color image using a custom colormap
mycolormap = [1 0 0; 0 1 0; 0 0 1; 1 1 1];
colormap(mycolormap)
mat = ones(50);
mat(1:25,1:25) = 4;
mat(1:25, 26:50) = 2;
mat(26:50, 26:50) = 3;
image(mat)
```

3) Write a function *numimage* that will receive two input arguments: a colormap matrix, and an integer “n”; the function will create an image that shows n “rings” of color, using the first n colors from the colormap. For example, if the function is called as follows:

```
>> cm = [0 0 0; 1 0 0; 0 1 0; 0 0 1; ...
1 1 0; 1 0 1; 0 1 1];
>> numimage(cm,5)
```

the following image will be created:

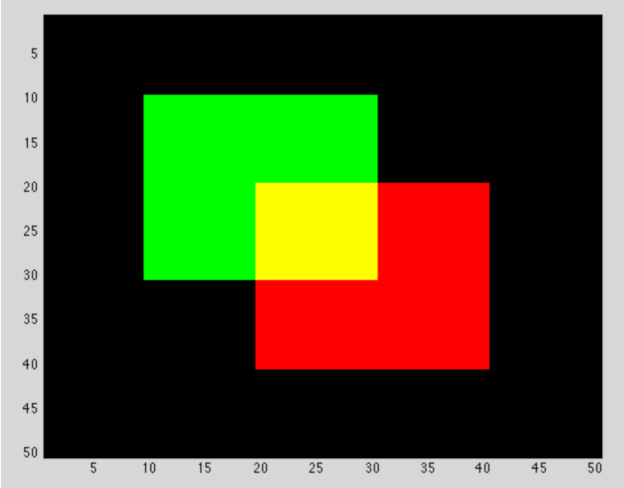


Each “ring” has the thickness of one pixel. In this case, since n was 5, the image shows the first 5 colors from the colormap: the outermost ring is the first color, the next ring is the second color, and the innermost pixel is the fifth color. Note that since n was 5, the image matrix is 9*9.

numimage.m

```
function numimage(cm, num)
colormap(cm)
cols = num * 2 - 1;
mat = ones(cols);
for i = 2:num
    cols = cols - 1;
    mat(i:cols,i:cols) = i;
end
image(mat)
end
```

4) Write a script that would produce the following 50 x 50 “image” using the RGB, or true color method (NOT the colormap method).



Ch13Ex4.m

```
imagemat = zeros(50,50,3);
imagemat(20:40,20:40,1) = 255;
imagemat(10:30,10:30,2) = 255;
imagemat = uint8(imagemat);
image(imagemat)
```

5) A script *rancolors* displays random colors in the Figure Window as shown in the Figure below. It starts with a variable *nColors* which is the

number of random colors to display (e.g., below this is 10). It then creates a colormap variable *mycolormap*, which has that many random colors, meaning that all three of the color components (red, green, and blue) are random real numbers in the range from 0 to 1. The script then displays these colors in an image in the Figure Window.

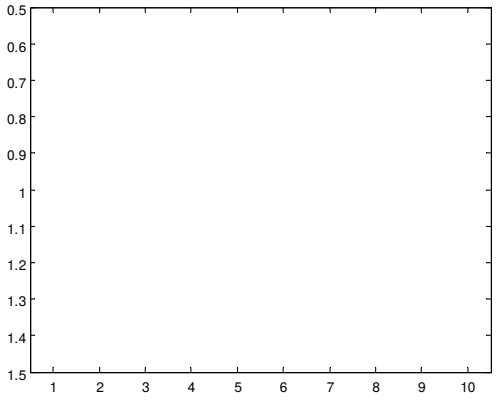
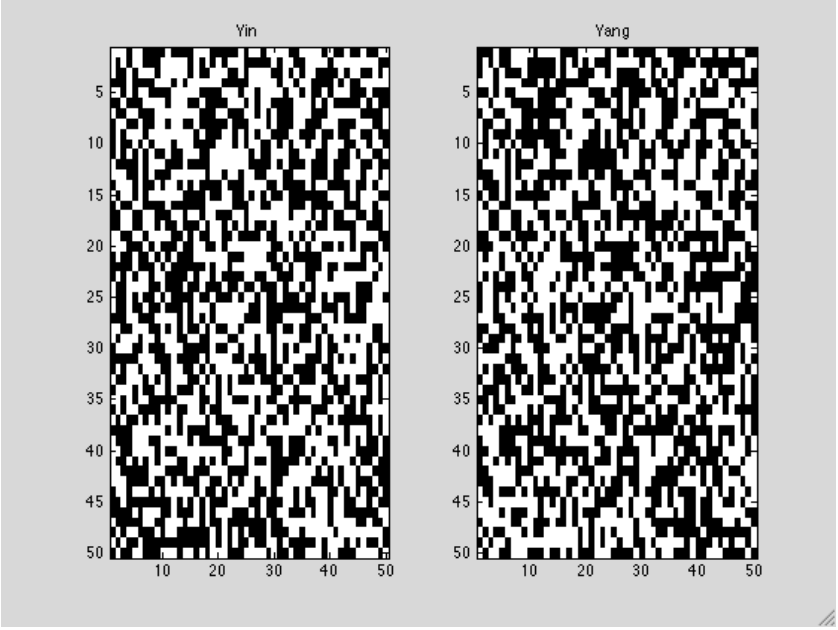


Figure Rainbow of random colors

rancolors.m

```
% Create n random colors and then display
% these in a row vector "image"

ncolors = 10;
mycolormap = rand(ncolors,3);
colormap(mycolormap)
vec = 1:ncolors;
image(vec)
```

6) Write a script that will create a colormap that just has two colors: white and black. The script will then create a 50x50 image matrix in which each element is randomly either white or black. In one Figure Window, display this image on the left. On the right, display another image matrix in which the colors have been reversed (all white pixels become black and vice versa). For example, the images might look like this (the axes are defaults; note the titles):



Do not use any loops or **if** statements.

Ch13Ex6.m

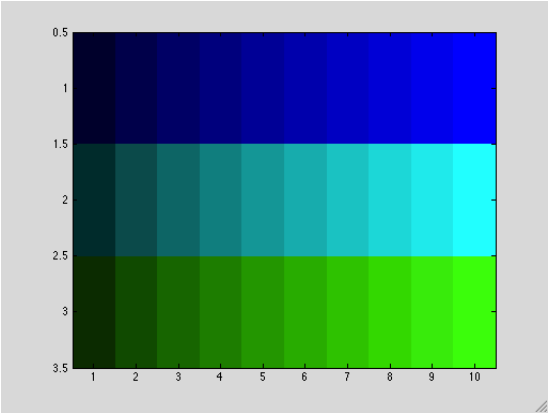
```
bwmap = [0 0 0; 1 1 1];
colormap(bwmap)

mat = randi(2,50);

subplot(1,2,1)
image(mat)
title('Yin')

subplot(1,2,2)

yang = ~(mat - 1);

image(yang)
title('Yang')
```

For the image matrix that you created, what would you expect the overall mean of the matrix elements to be?

1.5

7) Write a script that will show shades of green and blue. First, create a colormap that has 30 colors (ten blue, ten aqua, and then ten green). There is no red in any of the colors. The first ten rows of the colormap have no green, and the blue component iterates from 0.1 to 1 in steps

of 0.1. In the second ten rows, both the green and blue components iterate from 0.1 to 1 in steps of 0.1. In the last ten rows, there is no blue, but the green component iterates from 0.1 to 1 in steps of 0.1. Then, display all of the colors from this colormap in a 3 x 10 image matrix in which the blues are in the first row, aquas in the second, and greens in the third, as follows (the axes are the defaults). Do not use loops.



Ch13Ex7.m

```
colors = zeros(30,3);
vec = repmat(0.1:.1:1,1,2)'
colors(11:end,2) = vec;
colors(1:20,3) = vec;
colormap(colors)
mat = reshape(1:30,10,3)';
image(mat)
```

8) A part of an image is represented by an *n x n* matrix. After performing data compression and then data reconstruction techniques, the resulting matrix has values that are close to but not exactly equal to the original matrix. For example, the following *4 x 4* matrix variable *orig_im* represents a small part of a true color image, and *fin_im* represents the matrix after it has undergone data compression and then reconstruction.

```
orig_im =
   156    44   129    87
    18   158   118   102
    80    62   138    78
   155   150   241   105

fin_im =
   153    43   130    92
    16   152   118   102
    73    66   143    75
```

```
        152   155   247   114
```

Write a script that will simulate this by creating a square matrix of random integers, each in the range from 0 to 255. It will then modify this to create the new matrix by randomly adding or subtracting a random number (in a relatively small range, say 0 to 10) from every element in the original matrix. Then, calculate the average difference between the two matrices.

Ch13Ex8.m

```
% Simulates image compression and reconstruction, and
% Calculates average difference between matrices

orig_im = randi([0 255],4,4);

[r, c] = size(orig_im);

offval = randi([-7 9],r,c);
fin_im = orig_im + offval;

avediff = mean(mean(abs(offval)));
fprintf('The average difference is: %f\n',avediff)
```

9) It is sometimes difficult for the human eye to perceive the brightness of an object correctly. For example, in the Figure below, the middle of both images is the same color, and yet, because of the surrounding colors, the one on the left looks lighter than the one on the right.

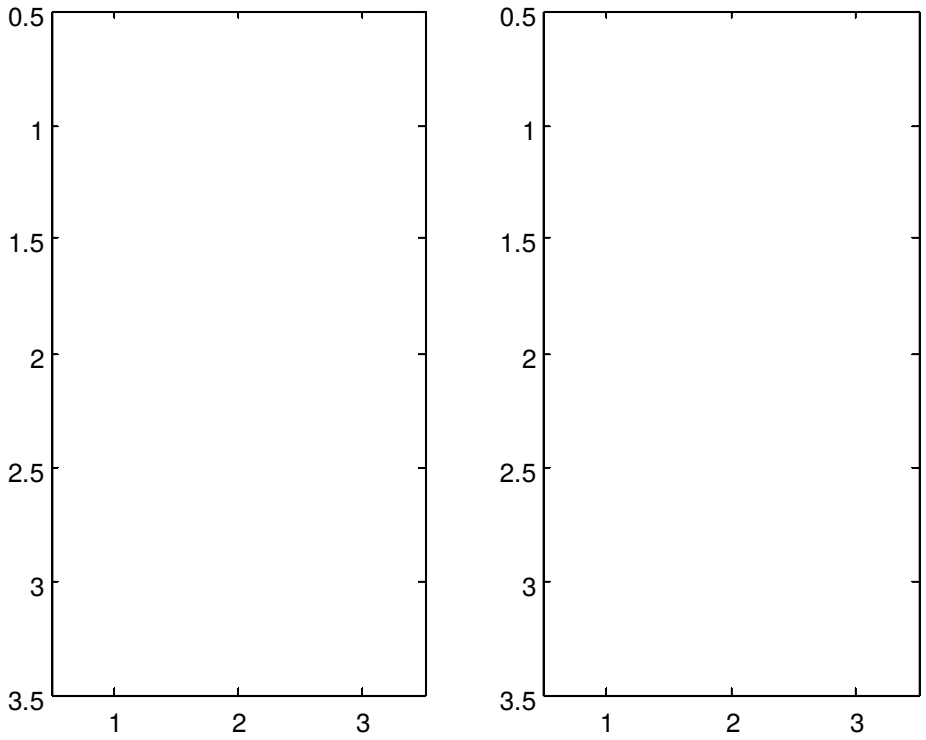


Figure Depiction of brightness perception

Write a script to generate a Figure Window similar to this one. Two *3 x 3* matrices were created. Use **subplot** to display both images side by side (the axes shown here are the defaults). Use the RGB method.

Ch13Ex9.m

```
% Produces a subplot to test perception
% of brightness

subplot(1,2,1)
mata = ones(3);
mata(2,2) = 15;
image(mata)
subplot(1,2,2)
matb = ones(3) + 32;
matb(2,2) = 15;
image(matb)
```

10) Put a JPEG file in your Current Folder and use **imread** to load it into a matrix. Calculate and print the **mean** separately of the red, green, and blue components in the matrix.

Ch13Ex10.m

```
im = imread('photo1.jpg');

[r c d] = size(im);

red = im(:,:,1);
green = im(:,:,2);
blue = im(:,:,3);

rmean = mean(mean(red));
gmean = mean(mean(green));
bmean = mean(mean(blue));

rstd = std(double(reshape(red,1,r*c)));
gstd = std(double(reshape(green,1,r*c)));
bstd = std(double(reshape(blue,1,r*c)));
fprintf('The mean of the red pixels is %.2f ',rmean)
fprintf('with a standard deviation of %.2f\n',rstd)
fprintf('The mean of the green pixels is %.2f ',gmean)
fprintf('with a standard deviation of %.2f\n',gstd)
fprintf('The mean of the blue pixels is %.2f ',bmean)
fprintf('with a standard deviation of %.2f\n',bstd)
```

11) Some image acquisition systems are not very accurate, and the result is ***noisy*** images. To see this effect, put a JPEG file in your Current Folder and use **imread** to load it. Then, create a new image matrix by randomly adding or subtracting a value *n* to every element in this matrix. Experiment with different values of *n*. Create a script that will use **subplot** to display both images side by side.

Ch13Ex11.m

```
% Make an image "noisy" and show both side by side

im = imread('photo1.jpg');
[r, c, d] = size(im);

im2 = im;
n = 50;

im2(:,:,1) = im2(:,:,1) + uint8(n*randi([-1 1],r,c));
im2(:,:,2) = im2(:,:,2) + uint8(n*randi([-1 1],r,c));
im2(:,:,3) = im2(:,:,3) + uint8(n*randi([-1 1],r,c));

subplot(1,2,1)
image(im)
subplot(1,2,2)
```

```
image(im2)
```

12) Put a JPEG file into your Current Folder. Type in the following script, using your own JPEG file name.

```
I1 = imread('xxx.jpg');
[r c h] = size(I1);
Inew(:,:,:) = I1(:,c:-1:1,:);
figure(1)
subplot(2,1,1)
image(I1);
subplot(2,1,2)
image(Inew);
```

Determine what the script does. Put comments into the script to explain it step-by-step.

Ch13Ex12.m

```
%Load image and get size
I1 = imread('photo1.jpg');
[r, c, h] = size(I1);
%Put columns in reverse order
Inew(:,:,:) = I1(:,c:-1:1,:);
%Plot original image and flipped image in a subplot
figure(1)
subplot(1,2,1)
image(I1);
subplot(1,2,2)
image(Inew);
```

13) Write a function that will create a simple GUI with one static text box near the middle of the Figure Window. Put your name in the string, and make the background color of the text box white.

Ch13Ex13.m

```
function Ch13Ex13
% Simple GUI with a static text box
% Format of call: Ch13Ex13
% Does not return any values

% Create the GUI but make it invisible for now while
%  it is being initialized
f = figure('Visible', 'off','color','white','Position',...
    [300, 400, 500, 325]);
htext = uicontrol('Style','text','Position', ...
    [200, 150, 100, 25], 'String','Hello, Kevin!',...
    'BackgroundColor','white');
```

```
% Put a name on it and move to the center of the screen
set(f,'Name','Simple GUI')
movegui(f,'center')

% Now the GUI is made visible
set(f,'Visible','on');
end
```

14) Write a function that will create a GUI with one editable text box near the middle of the Figure Window. Put your name in the string. The GUI should have a call-back function that prints the user's string twice, one under the other.

Ch13Ex14.m

```
function Ch13Ex14
% Simple GUI with an editable text box
% Prints string twice
% Format of call: Ch13Ex14
% Does not return any values

% Create the GUI but make it invisible for now while
%  it is being initialized
f = figure('Visible', 'off','color','white','Position',...
    [300, 400, 500, 325]);
hedit = uicontrol('Style','edit','Position', ...
    [150, 150, 200, 25],'Callback',@printname);

 % Put a name on it and move to the center of the screen
set(f,'Name','GUI with edit box')
movegui(f,'center')

% Now the GUI is made visible
set(f,'Visible','on');

    %Callback function for editable text field
    function printname(source,eventdata)
        set(hedit, 'Visible', 'off')
        str = get(hedit,'String');
        htxt1 = uicontrol('Style', 'text', 'Position', ...
            [150, 150, 200, 25], 'String', str,...
            'BackgroundColor', 'white');
        htxt2 = uicontrol('Style', 'text', 'Position', ...
            [150, 50, 200, 25], 'String', str,...
            'BackgroundColor', 'white');
    end
end
```

15) Fill in the callback function so that it gets the value of the slider, prints that value in the text box, and uses it to set the LineWidth of the plot (so, e.g., if the slider value is its maximum, the line width of the plot would be 5).

sliderlinewidth.m

```
function sliderlinewidth
f = figure('Visible', 'off','Position',[20,20,500,400]);
slhan = uicontrol('Style','slider','Units','Normalized',...
    'Position',[.3 .3 .4 .1],  ...
    'Min', 1, 'Max', 5,'Value',3,'Callback', @callbackfn);
slval = uicontrol('Style','text',...
    'Units','Normalized','Position', [.4 .1 .2 .1]);
axhan = axes('Units', 'Normalized','Position', [.3 .5 .4 .3]);
x = -2*pi:0.1:2*pi;
y = cos(x);
phan = plot(x,y)
set(f,'Visible','on');

    function callbackfn(source,eventdata)
       num=get(slhan, 'Value');
       set(slval, 'String',num2str(num))
       set(phan,'LineWidth',num)
    end
end
```

16) Write a function that creates a GUI to calculate the area of a rectangle. It should have edit text boxes for the length and width, and a push button that causes the area to be calculated and printed in a static text box.

Ch13Ex16.m

```
function Ch13Ex16
% GUI to calculate the area of a rectangle
% Format of call: Ch13Ex16
% Does not return any values

% Create the GUI but make it invisible for now while
%  it is being initialized
f = figure('Visible', 'off','color','white','Position',...
    [300, 400, 500, 325]);

hlengthtext = uicontrol('Style','text','Position', ...
    [190, 250, 50, 18],'String','Length:','HorizontalAlignment',...
```

```
    'right','BackgroundColor','white');

hlengthedit = uicontrol('Style','edit','Position', ...
    [250, 250, 50, 25]);

hwidthtext = uicontrol('Style','text','Position', ...
    [190, 200, 50, 18],'String','Width:','HorizontalAlignment',...
    'right','BackgroundColor','white');

hwidthedit = uicontrol('Style','edit','Position', ...
    [250, 200, 50, 25]);

hbutton = uicontrol('Style','pushbutton','Position',...
    [200, 150, 100, 25],'String','Calculate Area',...
    'Callback',@calcarea);

hareatext = uicontrol('Style','text','Position', ...
    [190, 100, 50, 18],'String','Area:','HorizontalAlignment',...
    'right','BackgroundColor','white');

hareaedit = uicontrol('Style','text','Position', ...
    [250, 100, 50, 18],'HorizontalAlignment',...
    'right','BackgroundColor','white');

% Put a name on it and move to the center of the screen
set(f,'Name','Rectangle Area Calculator')
movegui(f,'center')

% Now the GUI is made visible
set(f,'Visible','on');

    %Callback function for editable text field
    function calcarea(source,eventdata)
        len = str2num(get(hlengthedit,'String'));
        width = str2num(get(hwidthedit,'String'));
        area = len*width;
        set(hareaedit,'String',num2str(area))
    end

end
```

17) Write a function that creates a simple calculator with a GUI. The GUI should have two editable text boxes in which the user enters numbers. There should be four pushbuttons to show the four operations (+, -, *, /). When one of the four pushbuttons is pressed the type of operation should be shown in a static text box between the two

editable text boxes and the result of the operation should be displayed in a static text box.
If the user tries to divide by zero display an error message in a static text box.

guiCalculator.m

```
function guiCalculator
% Format of call: guiCalculator

f = figure('Visible','off','color','white',...
    'Position',[360 500 300 300]);

hop = uicontrol('Style','text','BackgroundColor','White',...
    'Position',[120 150 40 40]);
hequals = uicontrol('Style','text','BackgroundColor', 'White',...
    'Position',[200 150 40 40],'String','=','Visible','Off');
hresult = uicontrol('Style','text','BackgroundColor','White',...
    'Position',[240 150 40 40],'Visible','Off');
hfirst = uicontrol('Style','Edit','Position',[80 170 40 40]);
hsecond = uicontrol('Style','Edit','Position',[160 170 40 40]);

hadd = uicontrol('Style','pushbutton','Position',[45 50 50 50],...
    'String', '+','CallBack',@callbackfn);
hsub = uicontrol('Style','pushbutton','Position',[100 50 50 50],...
    'String', '-','CallBack',@callbackfn);
hmul = uicontrol('Style','pushbutton','Position',[155 50 50 50],...
    'String', '*','CallBack',@callbackfn);
hdiv = uicontrol('Style','pushbutton','Position',[210 50 50 50],...
    'String', '/','CallBack',@callbackfn);

hzero= uicontrol('Style','text','Position',[60 115 150 25],...
    'BackgroundColor','White','String',...
    'Cannot Divide by Zero','Visible','off');
set([hop hequals hresult hfirst hsecond hadd hsub hmul hdiv],...
        'Units','Normalized')
set(f,'Visible','On')

    function callbackfn(source,eventdata)
       firstnum = str2num(get(hfirst,'String'));
       secondnum = str2num(get(hsecond,'String'));
       set(hequals,'Visible','On')
       set(hzero,'Visible','off')
       switch source
           case hadd
               result = firstnum+secondnum;
               set(hop,'String','+')
               set(hresult,'String',num2str(result),'Visible','On')
           case hsub
               result = firstnum-secondnum;
```

```
                set(hop,'String','-')
                set(hresult,'String',num2str(result),'Visible','On')
            case hmul
                result = firstnum*secondnum;
                set(hop,'String','*')
                set(hresult,'String',num2str(result),'Visible','On')
            case hdiv
                if(secondnum == 0)
                    set(hzero,'Visible','on')
                else
                   result = firstnum/secondnum;
                   set(hop,'String','+')
                   set(hresult,'String',num2str(result),...
                     'Visible','On')
                end
        end
    end
end
```

18) Modify any example GUI to use the 'HorizontalAlignment' property to left-justify text within an edit text box.

guiWithLeftJustify.m

```
function guiWithLeftJustify
% Simple GUI with an edit box, using
% normalized units and left-justified text
% Format of call: guiWithLeftJustify
% Does not return any values

f = figure('Visible','off', 'Color', 'white', 'Units',...
    'Normalized','Position',[0.2 0.2 0.7 0.7], ...
    'Name','GUI using Normalized Units');
movegui(f, 'center')

% Create edit and static text boxes
hsttext = uicontrol('Style','text','BackgroundColor',...
    'white','Units','Normalized','Position',...
    [0.15 0.8 0.5 0.1], 'HorizontalAlignment','left',...
    'String','Enter your string here');

hedtext = uicontrol('Style','edit','BackgroundColor',...
    'white','Units','Normalized','Position',...
    [0.15 0.6 0.5 0.1], 'Callback', @callbackfn);
set(f, 'Visible', 'on')

   % Callback function
```

```
    function callbackfn(source, eventdata)
    set([hsttext, hedtext],'Visible','off');
    printstr = get(hedtext,'String');
    hstr = uicontrol('Style','text','BackgroundColor',...
        'white', 'Units', 'Normalized','Position',...
        [.2 .4 .6 .2], 'HorizontalAlignment','left',...
        'String',printstr,'FontSize',30,...
        'ForegroundColor','Red');
    set(hstr, 'Visible','on');
    end
end
```

19) 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). The formula is approximately

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

Write a GUI function that will display sliders for the temperature and wind speed. The GUI will calculate the WCF for the given values, and display the result in a text box. Choose appropriate minimum and maximum values for the two sliders.

guiWCF.m

```
function guiWCF
% GUI with Wind Chill Factor
% Format of call: guiWCF
% Does not return any values

% Create the GUI but make it invisible for now while
%  it is being initialized
f = figure('Visible', 'off','color','white','Position',...
    [300, 400, 500, 325]);

htempslider = uicontrol('Style','slider','Position', ...
    [150, 250, 200, 20],'Min',0,'Max',60,'Value',35,...
    'Callback',@update);

htemptext = uicontrol('Style','text','Position', ...
    [175, 275, 150, 18],'HorizontalAlignment',...
    'Center','BackgroundColor','white');

hvelslider = uicontrol('Style','slider','Position', ...
    [150, 150, 200, 20],'Min',0,'Max',30,'Value',15,...
    'Callback',@update);

hveltext = uicontrol('Style','text','Position', ...
    [175, 175, 150, 18],'HorizontalAlignment',...
```

```
    'Center','BackgroundColor','white');

hWCFtext = uicontrol('Style','text','Position', ...
    [175, 75, 150, 18],'HorizontalAlignment','Center',...
    'BackgroundColor','white');

% Put a name on it and move to the center of the screen
set(f,'Name','Simple GUI')
movegui(f,'center')
update()

% Now the GUI is made visible
set(f,'Visible','on');

    function update(source,eventdata)
        temp = get(htempslider,'Value');
        vel = get(hvelslider,'Value');
        set(htemptext,'String',...
            ['Temperature: ' num2str(round(temp)) ' F'])
        set(hveltext,'String',...
            ['Wind Velocity: ' num2str(round(vel)) ' MPH'])
        WCF =35.7+.6*round(temp)-35.7*(round(vel)).^.16+...
            0.43*round(temp)*(round(vel)).^(0.16);
        set(hWCFtext,'String',...
         ['Wind Chill Factor: ' num2str(round(WCF)) ' F'])
    end
end
```

20) Write a GUI function that will graphically demonstrate the difference between a **for** loop and a **while** loop. The function will have two push buttons: one that says ‘for’, and the other says ‘while’. There are two separate callback functions, one associated with each of the pushbuttons. The callback function associated with the ‘for’ button prints the integers 1 through 5, using **pause(1)** to pause for 1 second between each, and then prints ‘Done.’ The callback function associated with the ‘while’ button prints integers beginning with 1 and also pauses between each. This function, however, also has another pushbutton that says ‘mystery’ on it. This function continues printing integers until the ‘mystery’ button is pushed, and then it prints ‘Finally!’.

loopGUI.m

```
function loopGUI
f = figure('Visible', 'off','color','white',...
    'Position', [360, 500, 400,400]);
movegui(f,'center')
```

```
hbutton1 = uicontrol('Style','pushbutton','String',...
    'for', 'Position',[150,275,100,50], ...
    'Callback',@callbackfn1);
hbutton2 = uicontrol('Style','pushbutton','String',...
    'while', 'Position',[150,175,100,50], ...
    'Callback',@callbackfn2);
hstr = uicontrol('Style','text',...
    'BackgroundColor','white','Position',...
    [150,200,100,100],'FontSize',30,...
    'ForegroundColor','Red','Visible','off');
set(f,'Visible','on');

    function callbackfn1(source,eventdata)
        set([hbutton1 hbutton2],'Visible','off');
        set(hstr,'Visible','on')
        for i = 1:5
            set(hstr,'String', int2str(i));
            pause(1) % pause for 1 second
        end
        set(hstr,'String','Done!')

    end

    function callbackfn2(source,eventdata)
        set([hbutton1 hbutton2],'Visible','off');
        set(hstr,'Visible','on')
        cbmb = uicontrol('Style','pushbutton',...
            'String','mystery', 'Position',[300,50,50,50], ...
            'Callback',@cbfn,'Visible','on');
        done = false;
        i = 0;
        while ~done
            i = i + 1;
            set(hstr,'String',int2str(i));
            pause(1)
        end
        set(hstr,'String','Finally!')

        function cbfn(source,eventdata)
            done = true;
        end
    end
end
```

21) Write a function that will create a GUI in which there is a plot of cos(x). There should be two editable text boxes in which the user can enter the range for x.

guiCosPlot.m

```
function guiCosPlot
% Plots cos(x), allowing user to enter range
% Format of call: guiCosPlot
% Does not return any values

f = figure('Visible','off','Position',...
    [360, 500, 400, 400]);

% Edit boxes for min and max of x range

hmin = uicontrol('Style', 'edit','BackgroundColor',...
    'white', 'Position', [90, 285, 40, 40]);
hmax = uicontrol('Style', 'edit', 'BackgroundColor', ...
    'white', 'Position', [250, 285, 40, 40], ...
    'Callback', @callbackfn);

% Axis handle for plot

axhan = axes('Units', 'Pixels', 'Position', [100,50,200,200]);

set(f,'Name','Cos Plot Example')
movegui(f, 'center')
set([hmin, hmax], 'Units','Normalized')
set(f, 'Visible', 'on')

    % Callback function displays cos plot
    function callbackfn(source, eventdata)
        % Called by maximum edit box
        xmin = get(hmin, 'String');
        xmax = get(hmax, 'String');
        x = linspace(str2num(xmin),str2num(xmax));
        y = cos(x);
        plot(x,y)
    end
end
```

22) Write a function that will create a GUI in which there is a plot. Use a button group to allow the user to choose among several functions to plot.

guiChooseFnPlot.m

```
function guiChooseFnPlot
% Plots a function of x, allowing user to choose
%  function with a button group
```

```
% Format of call: guiChooseFnPlot
% Does not return any values

f = figure('Visible','off','Position',...
    [360, 500, 400, 400]);

% Create button group for function choice
grouph = uibuttongroup('Parent',f,'Units','Normalized',...
    'Position', [.3 .7 .3 .2], 'Title','Choose Function',...
    'SelectionChangeFcn', @whichfn);

sinh = uicontrol(grouph,'Style','radiobutton',...
    'String', 'sin', 'Units','Normalized',...
    'Position',[.2 .7 .4 .2]);
cosh = uicontrol(grouph,'Style','radiobutton',...
    'String','cos','Units','Normalized',...
    'Position',[.2 .4 .4 .2]);

set(grouph, 'SelectedObject', [])

% Axis handle for plot

axhan = axes('Units', 'Normalized', 'Position', [.2 .1 .5 .5]);

set(f,'Name','Choose Function Plot')
movegui(f, 'center')
set(f, 'Visible', 'on')

    function whichfn(source, eventdata)
        which = get(grouph, 'SelectedObject');
        x = -3 : 0.1 : 3;
        if which == sinh
            y = sin(x);
            plot(x,y)
        else
            y = cos(x);
            plot(x,y)
        end
    end
end
```

23) Write a GUI function that will create a **rectangle** object. The GUI has a slider on top that ranges from 2 to 10. The value of the slider determines the width of the **rectangle**. You will need to create axes for the rectangle. In the callback function, use **cla** to clear the children

from the current axes so that a thinner rectangle can be viewed.

guirectangle.m

```
function guirectangle

f = figure('Visible', 'off','Position', [360, 500, 400,400]);
minval = 2;
maxval = 10;
slhan = uicontrol('Style','slider','Position',
[140,280,100,50], ...
 'Min', minval, 'Max', maxval,'Value',minval,'Callback',
@callbackfn);
hmintext = uicontrol('Style','text','BackgroundColor',
'white', ...
    'Position', [90, 310, 40,15], 'String', num2str(minval));
hmaxtext = uicontrol('Style','text', 'BackgroundColor',
'white',...
    'Position', [250, 310, 40,15], 'String', num2str(maxval));
hsttext = uicontrol('Style','text','BackgroundColor', 'white',...
    'Position', [170,340,40,15],'Visible','off');
axhan = axes('Units', 'Pixels','Position', [100,50,200,200]);
set(f,'Name','Rectangle GUI')
movegui(f,'center')
set([slhan,hmintext,hmaxtext,hsttext,axhan],
'Units','normalized')
set(f,'Visible','on');

   function callbackfn(source,eventdata)
       num=get(slhan, 'Value');
       set(hsttext,'Visible','on','String',num2str(num))
       cla % deletes all children of current axes
       rh = rectangle('Position',[5,5,5,10]);
       axis([0 30 0 30])
       set(rh,'LineWidth',num)
    end
end
```

24) Write a GUI that displays an image in which all of the elements are the same color. Put 3 sliders in that allow the user to specify the amount of red, green, and blue in the image. Use the RGB method.

ColorGUI.m

```
function ColorGUI
 f = figure('Visible', 'off','Position',...
   [360, 500, 400,400]);
% Minimum and maximum values for sliders
minval = 0;
maxval = 255;
% Create the slider objects
```

```
rslhan = uicontrol('Style','slider','Units',...
    'Normalized','Position',[.1,.4,.25,.05], ...
    'Min', minval, 'Max', maxval,'SliderStep', [1 1],
'Callback', @callbackfn);
gslhan = uicontrol('Style','slider','Units',...
    'Normalized','Position',[.4,.4,.25,.05], ...
    'Min', minval, 'Max', maxval,'SliderStep', [1 1],
'Callback', @callbackfn);
bslhan = uicontrol('Style','slider','Units',...
    'Normalized','Position',[.7,.4,.25,.05], ...
    'Min', minval, 'Max', maxval,'SliderStep', [1 1],
'Callback', @callbackfn);

% Text boxes to show slider values
hredtext = uicontrol('Style','text','BackgroundColor',
'white', ...
    'Units','Normalized','Position', [.2,.3,.075,.025],...
    'Visible', 'off');
hgreentext = uicontrol('Style','text','BackgroundColor',
'white', ...
    'Units','Normalized','Position', [.525,.3,.075,.025],...
    'Visible', 'off');
hbluetext = uicontrol('Style','text','BackgroundColor',
'white', ...
    'Units','Normalized','Position', [.825,.3,.075,.025],...
    'Visible', 'off');
% Create axes handle for plot
axhan = axes('Units', 'Normalized','Position',
[.4,.6,.2,.2]);

movegui(f,'center')

set(f,'Visible','on');

% Call back function displays the current slider values &
% shows color
   function callbackfn(source,eventdata)
       rnum=get(rslhan, 'Value');
       gnum = get(gslhan, 'Value');
       bnum = get(bslhan, 'Value');
       set(hredtext,'Visible','on','String',num2str(rnum))
      set(hgreentext,'Visible','on','String',num2str(gnum))
       set(hbluetext,'Visible','on','String',num2str(bnum))
       mat = zeros(2,2,3);
       mat(:,:,1) = rnum;
       mat(:,:,2) = gnum;
       mat(:,:,3) = bnum;
```

```
        mat = uint8(mat);
        image(mat)
    end
end
```

25) Put two different JPEG files into your Current Folder. Read both into matrix variables. To superimpose the images, if the matrices are the same size, the elements can simply be added element-by-element. However, if they are not the same size, one method of handling this is to crop the larger matrix to be the same size as the smaller, and then add them. Write a script to do this.

Ch13Ex25.m

```
% Superimpose two images
% Crop one if necessary so they're the same size

im1 = imread('photo1.jpg');
im2 = imread('photo2.jpg');

[r1, c1, d1] = size(im1);
[r2, c2, d2] = size(im2);

%Check number of rows
if r1 > r2
    im1 = im1(1:r2,:,:);
elseif r1 < r2
    im2 = im2(1:r1,:,:);
end

%Check number of columns
if c1 > c2
    im1 = im1(:,1:c2,:);
elseif c1 < c2
    im2 = im2(:,1:c1,:);
end

[r1 c1 d1] = size(im1);
[r2 c2 d2] = size(im2);

%Superimpose
im3 = im1 + im2;
image(im3)
```

In a random walk, every time a "step" is taken, a direction is randomly chosen. Watching a random walk as it evolves, by viewing it as an

image, can be very entertaining. However, there are actually very practical applications of random walks; they can be used to simulate diverse events such as the spread of a forest fire or the growth of a dendritic crystal.

26) The following function simulates a "random walk," using a matrix to store the random walk as it progresses. To begin with all elements are initialized to 1. Then, the "middle" element is chosen to be the starting point for the random walk; a 2 is placed in that element. (Note: these numbers will eventually represent colors.) Then, from this starting point another element next to the current one is chosen randomly and the *color* stored in that element is incremented; this repeats until one of the edges of the matrix is reached. Every time an element is chosen for the next element, it is done randomly by either adding or subtracting one to/from each coordinate (*x* and *y*), or leaving it alone. The resulting matrix that is returned is an n by n matrix.

ranwalk.m

```
function walkmat = ranwalk(n)
walkmat = ones(n);
x = floor(n/2);
y = floor(n/2);
color = 2;
walkmat(x,y) = color;
while x ~= 1 && x ~= n && y ~= 1 && y ~= n
    x = x + randi([-1 1]);
    y = y + randi([-1 1]);
    color = color + 1;
    walkmat(x,y) = mod(color,65);
end
```

You are to write a script that will call this function twice (once passing 8 and once passing 100) and display the resulting matrices as images side-by-side. Your script must create a custom colormap that has 65 colors; the first is white and the rest are from the colormap **jet**. For example, the result may look like the Figure. (Note that with the 8 x 8 matrix the colors are not likely to get out of the blue range, but with 100 x 00 it cycles through all colors multiple times until an edge is reached):

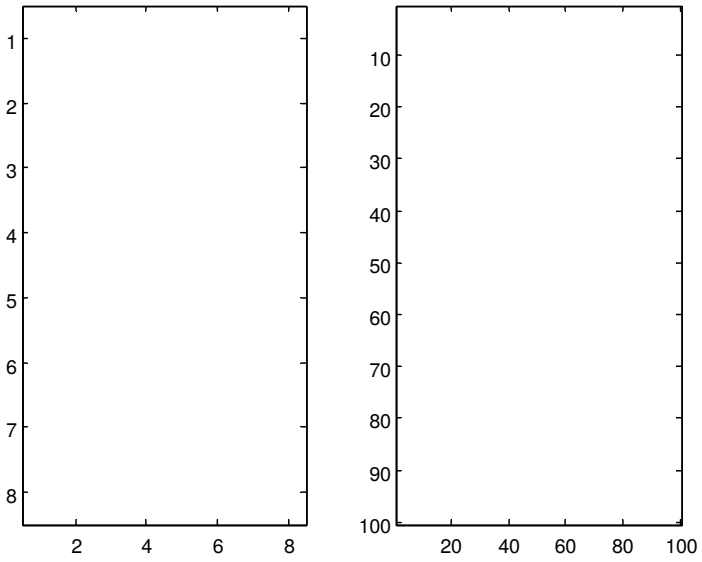


Figure Random walk

Ch13Ex26.m

```
% Show two examples of random walks side by side

cmap = colormap(jet);
mycolors = ones(65,3);
mycolors(2:end,:) = cmap;
colormap(mycolors)

subplot(1,2,1)
wmat = ranwalk(8);
image(wmat)

subplot(1,2,2)
wmat = ranwalk(100);
image(wmat)
```

27) Use App Designer to create a text editor. Create an app that has a large text box as seen in Figure 13.47. Under it there will be a slider that controls the font size of the text, and buttons to make the text bold and/or italic. Start by dragging a text box into the design area. Click on the box, and then in Design View look at the Edit Field (Text) Properties browser. By changing properties such as the style, Name, and Size, and then inspecting the code in Code View, you sit ametcan see what to change in the callback functions.

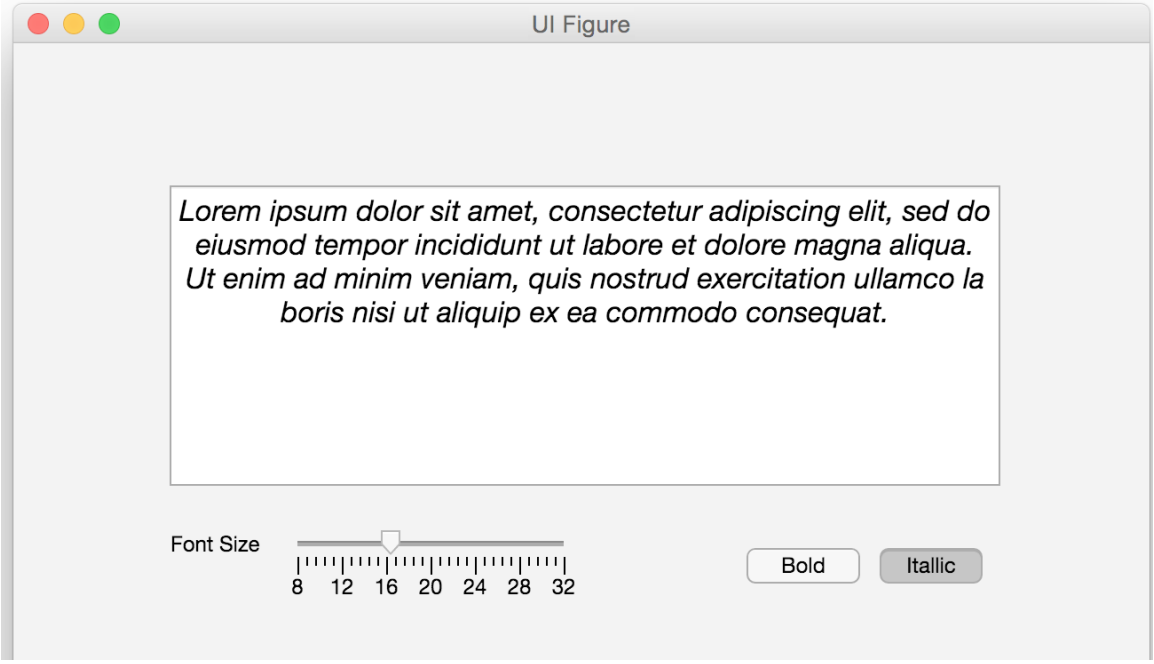


Figure 13.47 Text editor app

```
classdef TextEditor < matlab.apps.AppBase

    % Properties that correspond to app components

    properties (Access = public)

        UIFigure       matlab.ui.Figure            % UI Figure

        LabelSlider    matlab.ui.control.Label     % Font Size

        Slider         matlab.ui.control.Slider    % [8 32]

        Button         matlab.ui.control.Button    % Bold

        Button2        matlab.ui.control.Button    % Italic

        LabelEditField matlab.ui.control.Label

        EditField      matlab.ui.control.EditField % hello!

    end

    methods (Access = private)

        % Code that executes after component creation

        function startupFcn(app)
```

```
        end
        % Slider value changed function
        function FontCallBack(app)
            value = app.Slider.Value;
            app.EditField.FontSize = value;
        end
        % Button button pushed function
        function BoldButton(app)
            app.EditField.FontWeight = 'bold';
        end
        % Button2 button pushed function
        function ItalCallBack(app)
            app.EditField.FontAngle = 'italic';
        end
    end
    % App initialization and construction
    methods (Access = private)
        % Create UIFigure and components
        function createComponents(app)
            % Create UIFigure
            app.UIFigure = uifigure;
            app.UIFigure.Position = [100 100 640 480];
            app.UIFigure.Name = 'UI Figure';
            setAutoResize(app, app.UIFigure, true)
```

```
% Create LabelSlider
app.LabelSlider = uilabel(app.UIFigure);
app.LabelSlider.HorizontalAlignment = 'right';
app.LabelSlider.Position = [62.6875 101 51 15];
app.LabelSlider.Text = 'Font Size';
% Create Slider
app.Slider = uislider(app.UIFigure);
app.Slider.Limits = [8 32];
app.Slider.ValueChangedFcn = ...
    createCallbackFcn(app, @FontCallBack);
app.Slider.Position = [134.6875 107 150 3];
app.Slider.Value = 8;
% Create Button
app.Button = uibutton(app.UIFigure, 'push');
app.Button.ButtonPushedFcn = ...
    createCallbackFcn(app, @BoldButton);
app.Button.Position = [333 88 100 22];
app.Button.Text = 'Bold';
% Create Button2
app.Button2 = uibutton(app.UIFigure, 'push');
app.Button2.ButtonPushedFcn = ...
   createCallbackFcn(app, @ItalCallBack);
app.Button2.Position = [459 88 100 22];
app.Button2.Text = 'Italic';
% Create LabelEditField
```

```
            app.LabelEditField = uilabel(app.UIFigure);
            app.LabelEditField.HorizontalAlignment = 'right';
            app.LabelEditField.Position = [79.03125 379 20 15];
            app.LabelEditField.Text = '';
            % Create EditField
            app.EditField = uieditfield(app.UIFigure, 'text');
            app.EditField.Position = [114.03125 178 445 219];
            app.EditField.Value = 'hello!';
        end
    end
    methods (Access = public)
        % Construct app
        function app = TextEditor()
            % Create and configure components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            % Execute the startup function
            runStartupFcn(app, @startupFcn)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
```

```
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

28) Create a stoplight app as seen in Figure 13.48. There are two pushbuttons labeled ‘Stop’ and ‘Go’, and three lamps. When the ‘Go’ button is pushed, the green lamp is lit. When the ‘Stop’ button is pushed, the yellow lamp is lit briefly, and then the red lamp is lit.

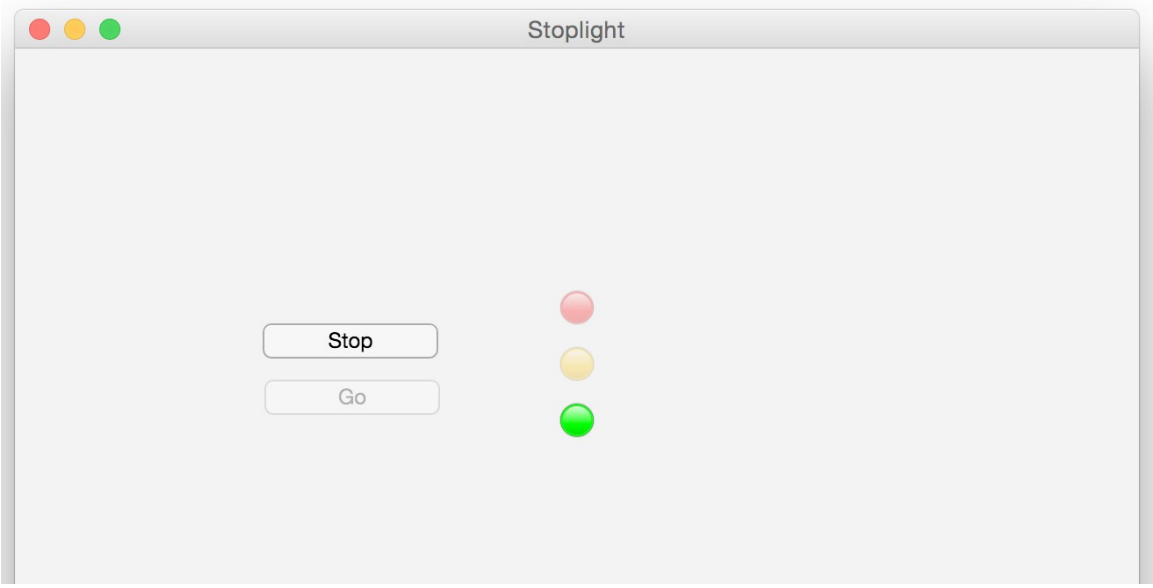


Figure 13.48 Stoplight app

```
classdef Stoplights < matlab.apps.AppBase
    % Properties that correspond to app components
    properties (Access = public)
        UIFigure   matlab.ui.Figure         % UI Figure
        Button     matlab.ui.control.Button % Stop
```

```
    Button2    matlab.ui.control.Button % Go
    LabelLamp  matlab.ui.control.Label
    Lamp       matlab.ui.control.Lamp
    LabelLamp2 matlab.ui.control.Label
    Lamp2      matlab.ui.control.Lamp
    LabelLamp3 matlab.ui.control.Label
    Lamp3      matlab.ui.control.Lamp
end
methods (Access = private)
    % Code that executes after component creation
    function startupFcn(app)

    end
    % Button2 button pushed function
    function GoButton(app)
        pause(1)
        app.Lamp.Visible = 'off';
        app.Lamp2.Visible = 'off';
        app.Lamp3.Visible = 'on';
    end
    % Button button pushed function
    function StopButton(app)
        app.Lamp3.Visible = 'off';
        app.Lamp2.Visible = 'on';
        pause(1)
```

```
        app.Lamp2.Visible = 'off';
        app.Lamp.Visible = 'on';
    end
end
% App initialization and construction
methods (Access = private)
    % Create UIFigure and components
    function createComponents(app)
        % Create UIFigure
        app.UIFigure = uifigure;
        app.UIFigure.Position = [100 100 640 480];
        app.UIFigure.Name = 'UI Figure';
        setAutoResize(app, app.UIFigure, true)
        % Create Button
        app.Button = uibutton(app.UIFigure, 'push');
        app.Button.ButtonPushedFcn = ...
             createCallbackFcn(app, @StopButton);
        app.Button.Position = [181 229 100 22];
        app.Button.Text = 'Stop';
        % Create Button2
        app.Button2 = uibutton(app.UIFigure, 'push');
        app.Button2.ButtonPushedFcn = ...
            createCallbackFcn(app, @GoButton);
        app.Button2.Position = [181 157 100 22];
        app.Button2.Text = 'Go';
```

```
% Create LabelLamp
app.LabelLamp = uilabel(app.UIFigure);
app.LabelLamp.Enable = 'off';
app.LabelLamp.HorizontalAlignment = 'right';
app.LabelLamp.Position = [387.03125 248 20 15];
app.LabelLamp.Text = '';
% Create Lamp
app.Lamp = uilamp(app.UIFigure);
app.Lamp.Visible = 'off';
app.Lamp.Position = [422.03125 245 20 20];
app.Lamp.Color = [1 0 0];
% Create LabelLamp2
app.LabelLamp2 = uilabel(app.UIFigure);
app.LabelLamp2.HorizontalAlignment = 'right';
app.LabelLamp2.Position = [387.03125 198 20 15];
app.LabelLamp2.Text = '';
% Create Lamp2
app.Lamp2 = uilamp(app.UIFigure);
app.Lamp2.Visible = 'off';
app.Lamp2.Position = [422.03125 195 20 20];
app.Lamp2.Color = [1 1 0];
% Create LabelLamp3
app.LabelLamp3 = uilabel(app.UIFigure);
app.LabelLamp3.HorizontalAlignment = 'right';
app.LabelLamp3.Position = [387.03125 140 20 15];
```

```
            app.LabelLamp3.Text = '';
            % Create Lamp3
            app.Lamp3 = uilamp(app.UIFigure);
            app.Lamp3.Visible = 'off';
            app.Lamp3.Position = [422.03125 137 20 20];
        end
    end
    methods (Access = public)
        % Construct app
        function app = Stoplights()
            % Create and configure components
            createComponents(app)
            % Register the app with App Designer
            registerApp(app, app.UIFigure)
            % Execute the startup function
            runStartupFcn(app, @startupFcn)
            if nargout == 0
                clear app
            end
        end
        % Code that executes before app deletion
        function delete(app)
            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
```

```
    end

end
```

29) Load two of the built-in MAT-file sound files (e.g. **gong** and **chirp)**. Store the sound vectors in two separate variables. Determine how to concatenate these so that the **sound** function will play one immediately followed by the other; fill in the blank here:

```
sound(       , 8192)
```

```
>> load gong
>> gy = y;
>> load chirp
>> sound([gy; y], Fs)
```

30) The following function *playsound* below plays one of the built-in sounds. The function has a cell array that stores the names. When the function is called, an integer is passed, which is an index into this cell array indicating the sound to be played. The default is 'train', so if the user passes an invalid index, the default is used. The appropriate MAT-file is loaded. If the user passes a second argument, it is the frequency at which the sound should be played (otherwise, the default frequency is used). The function prints what sound is about to be played and at which frequency, and then actually plays this sound. You are to fill in the rest of the following function. Here are examples of calling it (you can't hear it here, but the sound will be played!)

```
>> playsound(-4)
You are about to hear train at frequency 8192.0
>> playsound(2)
You are about to hear gong at frequency 8192.0
>> playsound(3,8000)
You are about to hear laughter at frequency 8000.0
```

playsound.m

```
function playsound(caind, varargin)
% This function plays a sound from a cell array
% of mat-file names
% Format playsound(index into cell array) or
%    playsound(index into cell array, frequency)
% Does not return any values

soundarray = {'chirp','gong','laughter','splat','train'};
if caind < 1 || caind > length(soundarray)
    caind = length(soundarray);
```

```
end
mysound = soundarray{caind};
eval(['load ' mysound])

% Fill in the rest
if nargin == 2
    Fs = varargin{1};
end
fprintf('You are about to hear %s at frequency %.1f\n',...
   mysound,Fs)
sound(y,Fs)
end
```

Chapter 14: Advanced Mathematics

Exercises

1) In a marble manufacturing plant, a quality control engineer randomly selects eight marbles from each of the two production lines and measures the diameter of each marble in millimeters. For the each data set here, determine the mean, median, mode, and standard deviation using built-in functions.

```
Prod. line A:15.94 15.98 15.94 16.16 15.86 15.86 15.90 15.88
Prod. line B:15.96 15.94 16.02 16.10 15.92 16.00 15.96  16.02
```

Suppose the desired diameter of the marbles is 16 mm. Based on the results you have, which production line is better in terms of meeting the specification? (Hint: think in terms of the mean and the standard deviation.)

Ch14Ex1.m

```
% Determine which marble production line has better
% quality

load marbles.dat
proda = marbles(1,:);
prodb = marbles(2,:);

fprintf('For production line A, the mean is %.2f,\n', mean(proda))
fprintf('the median is %.2f, the mode is %.2f,\n', ...
   median(proda), mode(proda))
fprintf(' and the standard deviation is %.2f\n', std(proda))
```

```
fprintf('For production line B, the mean is %.2f,\n', mean(prodb))
fprintf('the median is %.2f, the mode is %.2f,\n', ...
   median(prodb), mode(prodb))
fprintf(' and the standard deviation is %.2f\n', std(prodb))
```

Production line B seems better.

2) Write a function *mymin* that will receive any number of arguments, and will return the minimum. Note: the function is not receiving a vector; rather, all of the values are separate arguments.

mymin.m

```
function small = mymin(varargin)
% Receives any # of arguments, and returns the minimum
% Format of call: mymin(arguments)
% Returns the minimum of the arguments

n = nargin;
%Set initial value for the min
small = varargin{1};

%Loop through the remaining inputs and reassigning the min if a
smaller
%value is found
for i = 2:n
    if small > varargin{i}
        small = varargin{i};
    end
end
end
```

3) Write a script that will do the following. Create two vectors with 20 random integers in each; in one the integers should range from 1 to 5, and in the other, from 1 to 500 (inclusive). For each vector, would you expect the mean and median to be approximately the same? Would you expect the standard deviation of the two vectors to be approximately the same? Answer these questions, and then use the built-in functions to find the minimum, maximum, mean, median, standard deviation, and mode of each. Do a histogram for each in a subplot. Run the script a few times to see the variations.

Ch14Ex3.m

```
vec1 = randi([1 5],1,20);
vec2 = randi([1 500],1,20);

disp('For vector 1 with a range from 1-5:')
```

```
fprintf('The minimum is %d\n', min(vec1))
fprintf('The maximum is %d\n', max(vec1))
fprintf('The mean is %.1f\n', mean(vec1))
fprintf('The median is %.1f\n',median(vec1))
fprintf('The std deviation is %.1f\n', std(vec1))
fprintf('The mode is %.1f\n', mode(vec1))

disp('For vector 2 with a range from 1-500:')
fprintf('The minimum is %d\n', min(vec2))
fprintf('The maximum is %d\n', max(vec2))
fprintf('The mean is %.1f\n', mean(vec2))
fprintf('The median is %.1f\n',median(vec2))
fprintf('The std deviation is %.1f\n', std(vec2))
fprintf('The mode is %.1f\n', mode(vec2))

subplot(1,2,1)
hist(vec1)
subplot(1,2,2)
hist(vec2)
```

4) Write a function that will return the mean of the values in a vector, not including the minimum and maximum values. Assume that the values in the vector are unique. It is okay to use the built-in **mean** function. To test this, create a vector of 10 random integers, each in the range from 0 to 50, and pass this vector to the function.

outliers.m

```
function newmean = outliers(vec)
% Calculates the mean minus the minimum
% and maximum values
% Format of call: outliers(vector)
% Returns mean of vector except largest & smallest

%Find and remove the minimum value
[small, indlow] = min(vec);
vec(indlow) = [];

%Find and remove the maximum value
[large, indhigh] = max(vec);
vec(indhigh) = [];

%Calculate the mean of the rest of the vector
newmean = mean(vec);
end
```

```
>> vec = randi([0 50],1,10)
```

```
vec =
    18     5    39    19    12    20     4     6    48    48
>> ave = mean(vec)
ave =
   21.9000
>> outliers(vec)
ans =
   20.8750
>>
```

5) A moving average of a data set $x = \{x_1, x_2, x_3, x_4, \ldots, x_n\}$ is defined as a set of averages of subsets of the original data set. For example, a moving average of every two terms would be $1/2 *\{x_1+ x_2, x_2+ x_3, x_3 + x_4, \ldots, x_{n-1} + x_n\}$. Write a function that will receive a vector as an input argument, and will calculate and return the moving average of every two elements.

moveAve2.m

```
function moveave = moveAve2(vec)
% Calculates the moving average of every 2
% elements of a vector
% Format of call: moveAve2(vector)
% Returns the moving average of every 2 elements

firstpart = vec(1:end-1);
secondpart = vec(2:end);

moveave = 0.5 * (firstpart + secondpart);
end
```

Eliminating or reducing noise is an important aspect of any signal processing. For example, in image processing noise can blur an image. One method of handling this is called median filtering.

6) A median filter on a vector has a size, for example, a size of 3 means calculating the median of every three values in the vector. The first and last elements are left alone. Starting from the second element to the next-to-last element, every element of a vector vec(i) is replaced by the median of [vec(i-1) vec(i) vec(i+1)]. For example, if the signal vector is

```
signal = [5 11 4 2 6 8 5 9]
```

the median filter with a size of 3 is

```
medianFilter3 = [5 5 4 4 6 6 8 9]
```

Write a function to receive the original signal vector and return the median filtered vector.

medianFilter3.m

```
function outvec = medianFilter3(vec)
% Computes a median filter with a size of 3
% Format of call: medianFilter3(vector)
% Returns a median filter with size 3

outvec = vec;

for i = 2:length(vec) - 1
    outvec(i) = median([vec(i-1) vec(i) vec(i+1)]);
end
end
```

7) What is the difference between the mean and the median of a data set if there are only two values in it?

There is no difference.

8) A student missed one of four exams in a course and the professor decided to use the "average" of the other three grades for the missed exam grade. Which would be better for the student: the mean or the median if the three recorded grades were 99, 88, and 95? What if the grades were 99, 70, and 77?

```
>> median([99 88 95])          >> median([99 70 77])
ans =                          ans =
   95                             77

>> mean([99 88 95])            >> mean([99 70 77])
ans =                          ans =
   94                             82
```

9) Write a function *allparts* that will read in lists of part numbers for parts produced by two factories. These are contained in data files called *xyparts.dat* and *qzparts.dat*. The function will return a vector of all parts produced, in sorted order (with no repeats). For example, if the file *xyparts.dat* contains

```
123  145  111  333  456  102
```

and the file *qzparts.dat* contains

```
876  333  102  456  903  111
```

calling the function would return the following:

```
>> partslist = allparts
partslist =
   102   111   123   145   333   456   876   903
```

allparts.m

```
function outvec = allparts
% Reads in parts list from 2 data files
% Format of call: allparts or allparts()
% Returns a sorted list of all factory parts

load xyparts.dat
load qzparts.dat

outvec = union(xyparts,qzparts);
end
```

10) The set functions can be used with cell arrays of strings. Create two cell arrays to store (as strings) course numbers taken by two students. For example,

```
s1 = {'EC 101', 'CH 100', 'MA 115'};
s2 = {'CH 100', 'MA 112', 'BI 101'};
```

Use a set function to determine which courses the students have in common.

```
>> intersect(s1,s2)
ans =
    'CH 100'
```

11) A vector *v* is supposed to store unique random numbers. Use set functions to determine whether or not this is true.

```
>> v = randi([1 5], 1,8)
v =
     4     3     3     2     4     1     4     1
>> isequal(v, unique(v))
ans =
     0
>> v = 1:8;
>> isequal(v, unique(v))
ans =
     1
```

12) A program has a vector of structures that stores information on experimental data that has been collected. For each experiment, up to 10 data values were obtained. Each structure stores the number of data values for that experiment, and then the data values. The program is to calculate and print the average value for each experiment. Write a script to create some data in this format and print the averages.

Ch14Ex12.m

```
% create data structure
exper(3).numvals =  5;
exper(3).vals = 1:5;
exper(1).numvals = 1;
exper(1).vals = 33;
exper(2).numvals = 10;
exper(2).vals = [3:5 1.1:0.1:1.5 11];

for i = 1:length(exper)
    fprintf('The average value for experiment %d',i)
    fprintf(' was %.2f\n', mean(exper(i).vals))
end
```

13) Express the following polynomials as row vectors of coefficients:

$2x^3 - 3x^2 + x + 5$

$3x^4 + x^2 + 2x - 4$

```
>> poly1 = [2 -3 1 5];
>> poly2sym(poly1)
ans =
2*x^3 - 3*x^2 + x + 5
>> poly2 = [3 0 1 2 -4];
>> poly2sym(poly2)
ans =
3*x^4 + x^2 + 2*x - 4
```

14) Find the roots of the equation f(x) = 0 for the following function. Also, create *x* and *y* vectors and plot this function in the range from -3 to 3 to visualize the solution.

$f(x) = 3x^2 - 2x - 5$

```
>> x = sym('x');
>> eqn = 3*x^2 - 2*x - 5;
>> root = solve(eqn)
root =
  -1
 5/3
>> ezplot(eqn,[-3,3])
```

15) Evaluate the polynomial expression $3x^3 + 4x^2 + 2x - 2$ at x = 4, x = 6, and x = 8.

```
>> expr = [3 4 2 -2];
>> polyval(expr, 4:2:8)
```

16) What is a danger of extrapolation?

```
If you go too far from the range of data points, it is
likely that extrapolated values will be meaningless.
```

17) Write a script that will generate a vector of 10 random integers, each in the inclusive range from 0 to 100. If the integers are evenly distributed in this range, then when arranged in order from lowest to highest, they should fall on a straight line. To test this, fit a straight line through the points and plot both the points and the line with a legend.

Ch14Ex17.m

```
% Tests to see whether random integers are fairly evenly
% distributed in a range by fitting a straight line to them

rnums = randi([0 100],1,10);
srtd = sort(rnums);
x=1:10;

coefs=polyfit(x,srtd,1);
curve = polyval(coefs,x);
plot(x,srtd,'o',x,curve)
title('Straight line through random points')
legend('random points','straight line')
```

18) Write a function that will receive data points in the form of x and y vectors. If the lengths of the vectors are not the same, then they can't represent data points so an error message should be printed. Otherwise, the function will fit a polynomial of a random degree through the points, and will plot the points and the resulting curve with a title specifying the degree of the polynomial. The degree of the polynomial must be less than the number of data points, n, so the function must generate a random integer in the range from 1 to n-1 for the polynomial degree.

ranCurveFit.m

```
function ranCurveFit(x,y)
% Uses x,y input data vectors, performs a curve fit with a random
% polynomial degree. Function terminates if input vectors
% have different lengths.
% Format of call: ranCurveFit(x,y)
% Does not return any values

if length(x) ~= length(y)
    disp('Error! x and y must have the same length')
```

```
else
    n = randi([1,length(x)-1]);
    coefs = polyfit(x,y,n);
    curve = polyval(coefs,x);
    plot(x,y,'ko',x,curve)
    title(sprintf('Polynomial degree: %d',n))
end
end
```

19) Write a function *mirror* that will receive one input vector consisting of y coordinates of data points. The function will fit a second order polynomial through the points. The function will plot, on one graph, the original data points (using green *s), the curve (using blue and with enough points so that it is very smooth) and also “mirror image” points (in red *s). The “mirror image” points are, for every x coordinate, equidistant from the curve as the original data point. So, if the original data point is 2 above the curve, the mirror image point will be 2 below it.

mirror.m

```
function mirror(y)
x = 1:length(y);
p = polyfit(x,y,2);
longx = 1:0.1:length(x);
curve = polyval(p, longx);
ests = polyval(p,x);
oppests = ests + (ests - y);
plot(x,y,'g*',longx,curve,x,oppests,'r*')
end
```

Data on the flow of water in rivers and streams is of great interest to civil engineers, who design bridges, and to environmental engineers, who are concerned with the environmental impact of catastrophic events such as flooding.

20) The Mystical River’s water flow rate on a particular day is shown in the table below. The time is measured in hours and the water flow rate is measured in cubic feet per second. Write a script that will fit polynomials of degree 3 and 4 to the data and create a subplot for the two polynomials. Plot also the original data as black circles in both plots. The titles for the subplots should include the degree of the fitted polynomial. Also, include appropriate x and y labels for the plots.

Time	0	3	6	9	12	15	18	21	24

Flow Rate	800	980	1090	1520	1920	1670	1440	1380	1300

Ch14Ex20.m

```
% Plot the water flow rate for the Mystical River one day
% Fit polynomials of order 3 and 4 through the points and plot

time = 0:3:24;
flows = [800 980 1090 1520 1920 1670 1440 1380 1300];

subplot(1,2,1)
curve = polyfit(time,flows,3);
y3 = polyval(curve,time);
plot(time,flows,'ko',time,y3)
xlabel('Time')
ylabel('Flow rate')
title('Order 3 polynomial')

subplot(1,2,2)
curve = polyfit(time,flows,4);
y4 = polyval(curve,time);
plot(time,flows,'ko',time,y4)
xlabel('Time')
ylabel('Flow rate')
title('Order 4 polynomial')
```

21) Write a function that will receive x and y vectors representing data points. The function will create, in one Figure Window, a plot showing these data points as circles and also in the top part a second-order polynomial that best fits these points and on the bottom a third-order polynomial. The top plot will have a line width of 3 and will be a gray color. The bottom plot will be blue, and have a line width of 2. For example, the Figure Window might look like this.

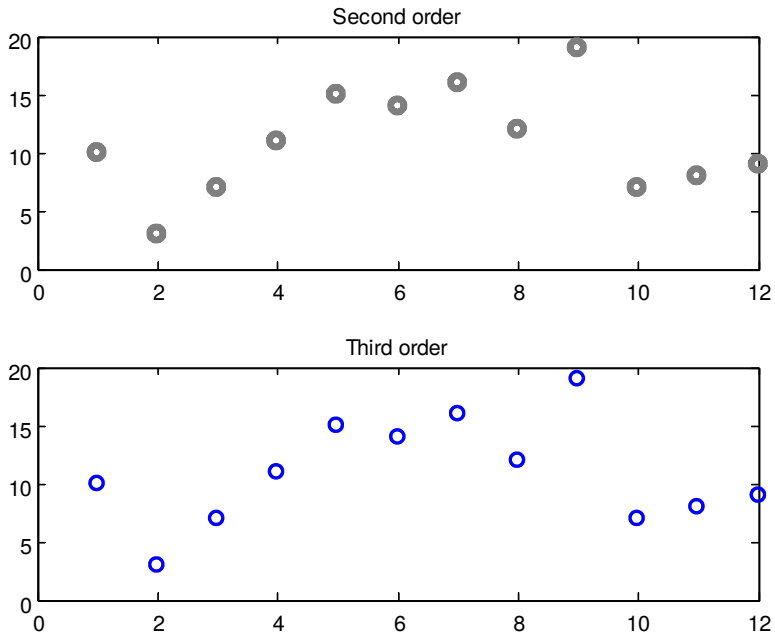


Figure Subplot of second and third order polynomials with different line properties

The axes are the defaults. Note that changing the line width also changes the size of the circles for the data points. You do not need to use a loop.

fitLineWidth.m

```
function fitLineWidth(x,y)
% Plots data points with order 2 and 3
% polynomials fit through them, with line
% widths and color specified
% Format of call: fitLineWidth(x,y)
% Does not return any values

subplot(2,1,1)
coefs = polyfit(x,y,2);
curve = polyval(coefs,x);
plot(x,y,'o',x,curve,'LineWidth',3,'Color',[0.5 0.5 0.5])
title('Second order')
subplot(2,1,2)
coefs = polyfit(x,y,3);
curve = polyval(coefs,x);
plot(x,y,'o',x,curve,'LineWidth',2, 'Color', [0 0 1])
title('Third order')
end
```

22) Store the following complex numbers in variables, and print them in the form a + bi.

3-2i

$\sqrt{-3}$

```
>> z1 = 3-2*i;
>> z2 = sqrt(-3);
>> fprintf('z1 = %.2f + %.2fi\n',real(z1),imag(z1))
z1 = 3.00 + -2.00i
>> fprintf('z2 = %.2f + %.2fi\n',real(z2),imag(z2))
z2 = 0.00 + 1.73i
```

23) Create the following complex variables
c1 = 2 -4i;
c2 = 5+3i;
Perform the following operations on them:

- add them
- multiply them
- get the complex conjugate and magnitude of each
- put them in polar form

Ch14Ex23.m

```
% Create complex variables and perform
% several operations

c1 = 2-4*i;
c2 = 5+3*i;
%Sum
c1+c2
%Product
c1*c2
%Complex conjugates
conj(c1)
conj(c2)
%Magnitude
abs(c1)
abs(c2)
%Polar form
r = abs(c1)
theta = angle(c1)
r = abs(c2)
theta = angle(c1)
```

24) Represent the expression $z^3 - 2z^2 + 3 - 5i$ as a row vector of coefficients, and store this in a variable *compoly*. Use the **roots**

function to solve $z^3 - 2z^2 + 3 - 5i = 0$. Also, find the value of *compoly* when z = 2 using **polyval**.

```
>> compoly = [1 -2 3-5i];
>> croots = roots(compoly)
croots =
   2.3010 + 1.9216i
  -0.3010 - 1.9216i
>> val = polyval(compoly,2)
val =
   3.0000 - 5.0000i
```

25) What is the value of the trace of an *n x n* identity matrix?

The trace is n.

26) For an *m x n* matrix, what are the dimensions of its transpose?

n x m

27) What is the transpose of a diagonal matrix A?

A

28) When is a square matrix both an upper triangular and lower triangular matrix?

When it is a diagonal matrix

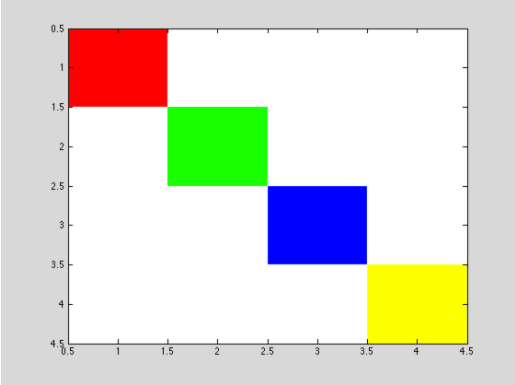
29) Is the transpose of an upper triangular matrix also upper triangular? If not, how would you characterize it?

No, it is lower triangular

30) Given the following colormap matrix:

```
mycmap = [1 1 1; 1 0 0; 0 1 0; 0 0 1; 1 1 0];
```

Write code that will generate the following 4x4 “image” matrix, using the colormap method:



Note: the axes are the defaults, and you only need 3 lines of code to accomplish this.

```
colormap(mycmap)
immat = diag(1:4) + 1;
image(immat)
```

31) Write a function *myupp* that will receive an integer argument *n*, and will return an *n x n* upper triangular matrix of random integers.

myupp.m

```
function mat = myupp(n)
% Creates an n x n upper triangular matrix
% of random integers
% Format of call: myupp(n)
% Returns upper triangular matrix

% Creates the matrix
mat = randi([-10,10],n,n);

% Programming method
for i = 1:n
    for j = 1:i-1
        mat(i,j) = 0;
    end
end
end
```

32) Analyzing electric circuits can be accomplished by solving sets of equations. For a particular circuit, the voltages V_1, V_2, and V_3 are found through the system:

$V_1 = 5$

$-6V_1 + 10V_2 - 3V_3 = 0$

$-V_2 + 51V_3 = 0$

Put these equations in matrix form and solve in MATLAB.

```
>> A = [1 0 0; -6 10 -3; 0 -1 51];
>> b = [5;0;0];
>> v = inv(A) * b

v =

    5.0000
    3.0178
    0.0592
```

33) Re-write the following system of equations in matrix form:

$$\begin{aligned} 4x_1 - x_2 + 3x_4 &= 10 \\ -2x_1 + 3x_2 + x_3 - 5x_4 &= -3 \\ x_1 + x_2 - x_3 + 2x_4 &= 2 \\ 3x_1 + 2x_2 - 4x_3 &= 4 \end{aligned}$$

Set it up in MATLAB and use any method to solve.

$$\begin{bmatrix} 4 & -1 & 0 & 3 \\ -2 & 3 & 1 & -5 \\ 1 & 1 & -1 & 2 \\ 3 & 2 & -4 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ -3 \\ 2 \\ 4 \end{bmatrix}$$

```
>> A = [4 -1 0 3;
    -2 3 1 -5;
    1 1 -1 2;
    3 2 -4 0];
>> b = [10 -3 2 4]';
>> x = inv(A)*b
```

34) Solve the simultaneous equations $x - y = 2$ and $x^2 + y = 0$ using **solve**. Plot the corresponding functions, $y = x-2$ and $y = -x^2$, on the same graph with an x range from -5 to 5.

Ch14Ex34.m

```
% Solve 2 simultaneous equations using solve, and plot

syms x y
answer = solve('x - y = 2', 'x^2 + y = 0',x,y);
x = answer.x
y = answer.y
ezplot('y = x - 2',[-5,5])
hold on
ezplot('y = -x^2',[-5,5])
```

35) For the following set of equations,

$$2x_1 + 2x_2 + x_3 = 2$$
$$x_2 + 2x_3 = 1$$
$$x_1 + x_2 + 3x_3 = 3$$

write it in symbolic form and solve using the **solve** function. From the symbolic solution, create a vector of the numerical (**double**) equivalents.

```
>> solve('2*x+2*y+z=2','y+2*z=1','x+y+3*z=3')
ans =
    x: [1x1 sym]
    y: [1x1 sym]
    z: [1x1 sym]

>> double([ans.x ans.y ans.z])
ans =
    1.2000   -0.6000    0.8000
```

36) The reproduction of cells in a bacterial colony is important for many environmental engineering applications such as wastewater treatments. The formula

$$\log(N) = \log(N_0) + t/T \log(2)$$

can be used to simulate this, where N_0 is the original population, N is the population at time t, and T is the time it takes for the population to double. Use the **solve** function to determine the following: if $N_0 = 10^2$, $N = 10^8$, and t = 8 hours, what will be the doubling time T? Use **double** to get your result in hours.

```
>> No = 10^2;
>> N = 10^8;
>> t = 8;
>> T = sym('T');
>> eqn = log(No) + (t/T)*log(2) - log(N);
>> double(solve(eqn))
ans =
    0.4014
```

37) Using the symbolic function **int**, find the indefinite integral of the function $4x^2 + 3$, and the definite integral of this function from x = -1 to x = 3. Also, approximate this using the **trapz** function.

Ch14Ex37.m

```
% Find integrals
```

```
x = sym('x');
f = 4*x^2 + 3;
%Indefinite integral
int(f)
%Definite integral from x = -1 to x = 3
int(f,-1,3)
%Approximation
f = sym2poly(f);
x = -1:0.1:3;
y = polyval(f,x);
```

38) Use the **quad** function to approximate the area under the curve $4x^2 + 3$ from -1 to 3. First, create an anonymous function and pass its handle to the **quad** function.

```
>> fun = @(x) 4*x.^2 + 3;
>> quad(fun,-1,3)
```

39) Use the **polyder** function to find the derivative of $2x^3 - x^2 + 4x - 5$.

```
>> der = polyder([2 -1 4 5])
der =
     6    -2     4
>> poly2sym(der)
ans =
6*x^2 - 2*x + 4
```

40) Examine the motion, or *trajectory*, of a *projectile* moving through the air. Assume that it has an initial height of 0, and neglect the air resistance for simplicity. The projectile has an initial velocity v_0, an angle of departure θ_0, and is subject to the gravity constant g = 9.81m/s^2. The position of the projectile is given by x and y coordinates, where the origin is the initial position of the projectile at time t = 0. The total horizontal distance that the projectile travels is called its *range* (the point at which it hits the ground), and the highest peak (or vertical distance) is called its *apex*. Equations for the trajectory can be given in terms of the time t or in terms of x and y. The position of the projectile at any time t is given by:

$$x = v_0 \cos(\theta_0)\, t$$

$$y = v_0 \sin(\theta_0)\, t - \tfrac{1}{2}\, g\, t^2$$

For a given initial velocity v_0, and angle of departure θ_0, describe the motion of the projectile by writing a script to answer the following:

- What is the range?
- Plot the position of the projectile at suitable x values
- Plot the height versus time.
- How long does it take to reach its apex?

Ch14Ex40.m

```
v0 = 33;
theta0 = pi/4;
t = 0;
g = 9.81;
xvec = [];
yvec = [];

[x, y] = findcoords(v0,theta0,t);

while y >= 0
    xvec = [xvec x];
    yvec = [yvec y];
    plot(x,y)
    hold on
    t = t + 0.01;
    [x, y] = findcoords(v0,theta0,t);
end

fprintf('The range is %.1f\n', xvec(end))
fprintf('The apex is %.1f\n', max(yvec))
```

findcoords.m

```
function [x, y] = findcoords(v0, theta0, t)
g = 9.81;

x = v0 * cos(theta0) * t;
y = v0 * sin(theta0) * t - 0.5 * g * t * t;
end
```

41) Write a GUI function that creates four random points. Radio buttons are used to choose the order of a polynomial to fit through the points. The points are plotted along with the chosen curve.

buttonGUI.m

```
function buttonGUI

```

```
f = figure('Visible', 'off','Position',...
   [360, 500, 400,400]);

grouph = uibuttongroup('Parent',f,'Units','Normalized',...
    'Position',[.3 .6 .4 .3], 'Title','Choose Order',...
    'SelectionChangeFcn',@whattodo);

but1 = uicontrol(grouph,'Style','radiobutton',...
    'String','First','Units','Normalized',...
    'Position', [.2 .8 .4 .2]);

but2 = uicontrol(grouph, 'Style','radiobutton',...
    'String','Second','Units','Normalized',...
    'Position',[.2 .5 .4 .2]);

but3 = uicontrol(grouph, 'Style','radiobutton',...
    'String','Third','Units','Normalized',...
    'Position',[.2 .2 .4 .2]);

axhan = axes('Units','Normalized','Position',...
           [.2,.2,.7,.3]);
x = 1:4;
y = randi(10, [1,4]);
lotx = 1: 0.2: 4;

set(grouph,'SelectedObject',[])

set(f,'Name','Exam GUI')
movegui(f,'center')
set(f,'Visible','on');

    function whattodo(source, eventdata)

which = get(grouph,'SelectedObject');

    if which == but1
        coefs = polyfit(x,y,1);
    elseif which == but2
        coefs = polyfit(x,y,2);
    else
        coefs = polyfit(x,y,3);
    end

    curve = polyval(coefs,lotx);

    plot(x,y,'ro',lotx,curve)
    end
```

```
end
```