Commands for Managing a Session

Command	Purpose
clc	Clears command window.
clear	Removes variables from memory.
exist	Checks for existence of file or variable.
global	Declares variables to be global.
help	Searches for a help topic.
lookfor	Searches help entries for a keyword.
quit	Stops MATLAB.
who	Lists current variables.
whos	Lists current variables (long display).

Commands for Working with the System

Command	Purpose
cd	Changes current directory.
date	Displays current date.
delete	Deletes a file.
diary	Switches on/off diary file recording.
dir	Lists all files in current directory.
load	Loads workspace variables from a file.

path	Displays search path.
pwd	Displays current directory.
save	Saves workspace variables in a file.
type	Displays contents of a file.
what	Lists all MATLAB files in the current directory.
wklread	Reads .wk1 spreadsheet file.

Input and Output Command

Command	Purpose
disp	Displays contents of an array or string.
fscanf	Read formatted data from a file.
format	Controls screen-display format.
fprintf	Performs formatted writes to screen or file.
input	Displays prompts and waits for input.
;	Suppresses screen printing.

The **fscanf** and **fprintf** commands behave like C scanf and printf functions

Format Code	Purpose
%s	Format as a string.
%d	Format as an integer.
%f	Format as a floating point value.

%e	Format as a floating point value in scientific notation.
%g	Format in the most compact form: %f or %e.
\n	Insert a new line in the output string.
\t	Insert a tab in the output string.

The format function has the following forms used for numeric display −

Format Function	Display up to
format short	Four decimal digits (default).
format long	16 decimal digits.
format short e	Five digits plus exponent.
format long e	16 digits plus exponents.
format bank	Two decimal digits.
format +	Positive, negative, or zero.
format rat	Rational approximation.
format compact	Suppresses some line feeds.
format loose	Resets to less compact display mode.

Vector, Matrix and Array Commands

Command	**Purpose**
cat	Concatenates arrays.

find	Finds indices of nonzero elements.
length	Computes number of elements.
linspace	Creates regularly spaced vector.
logspace	Creates logarithmically spaced vector.
max	Returns largest element.
min	Returns smallest element.
prod	Product of each column.
reshape	Changes size.
size	Computes array size.
sort	Sorts each column.
sum	Sums each column.
eye	Creates an identity matrix.
ones	Creates an array of ones.

zeros	Creates an array of zeros.
cross	Computes matrix cross products.
dot	Computes matrix dot products.
det	Computes determinant of an array.
inv	Computes inverse of a matrix.
pinv	Computes pseudoinverse of a matrix.
rank	Computes rank of a matrix.
rref	Computes reduced row echelon form.
cell	Creates cell array.
celldisp	Displays cell array.
cellplot	Displays graphical representation of cell array.
num2cell	Converts numeric array to cell array.
deal	Matches input and output lists.

iscell	Identifies cell array.

Plotting Commands

MATLAB provides numerous commands for plotting graphs. The following table shows some of the commonly used commands for plotting −

Command	Purpose
axis	Sets axis limits.
fplot	Intelligent plotting of functions.
grid	Displays gridlines.
plot	Generates xy plot.
print	Prints plot or saves plot to a file.
title	Puts text at top of plot.
xlabel	Adds text label to x-axis.
ylabel	Adds text label to y-axis.
axes	Creates axes objects.
close	Closes the current plot.
close all	Closes all plots.
figure	Opens a new figure window.
gtext	Enables label placement by mouse.
hold	Freezes current plot.
legend	Legend placement by mouse.

refresh	Redraws current figure window.
set	Specifies properties of objects such as axes.
subplot	Creates plots in subwindows.
text	Places string in figure.
bar	Creates bar chart.
loglog	Creates log-log plot.
polar	Creates polar plot.
semilogx	Creates semilog plot. (logarithmic abscissa).
semilogy	Creates semilog plot. (logarithmic ordinate).
stairs	Creates stairs plot.
stem	Creates stem plot.

Creating and Running Script File

- Using the command prompt
- Using the IDE

```
edit
Or
edit <filename>
```

Let us create a folder named progs. Type the following commands at the command prompt (>>) −

```
mkdir progs    % create directory progs under default directory

chdir progs    % changing the current directory to progs
```

```
edit  prog1.m  % creating an m file named prog1.m
```

Type the following code in the editor −

```
NoOfStudents = 6000;
TeachingStaff = 150;
NonTeachingStaff = 20;

Total = NoOfStudents + TeachingStaff ...
+ NonTeachingStaff;
disp(Total);
```

A. Basic Syntax and Command line exercise:

1. Create a vector of the even whole numbers between 31 and 75.

   x = 32:2:75

2. Let x = [2 5 1 6].

   a. Add 16 to each element
   b. Add 3 to just the odd-index elements
   c. Compute the square root of each element
   d. Compute the square of each element

Ans:

x = [2 5 1 6]

a = x + 16

b = x(1:2:end) + 3

c = sqrt(x) **or** c = x.^(0.5)

d = x.^2 **or** d = x.*x

3. Let x = [3 2 6 8]' and y = [4 1 3 5]' (NB. x and y should be column vectors).

a. Add the sum of the elements in x to y

b. Raise each element of x to the power specified by the corresponding element in y.

c. Divide each element of y by the corresponding element in x

d. Multiply each element in x by the corresponding element in y, calling the result "z".

e. Add up the elements in z and assign the result to a variable called "w".

f. Compute x'*y - w and interpret the result

```
x = [3 2 6 8]', y = [4 1 3 5]'
    a = y + sum(x)
    b = x.^y
    c = y./x
    z = x.*y
    w = sum(z)
    x'*y - w   (same thing)
```

4. Evaluate the following MATLAB expressions by hand and use MATLAB to check the answers

a. 2 / 2 * 3

b. 6 - 2 / 5 + 7 ^ 2 - 1

c. 10 / 2 \ 5 - 3 + 2 * 4

d. 3 ^ 2 / 4

e. 3 ^ 2 ^ 2

f. 2 + round(6 / 9 + 3 * 2) / 2 - 3

g. 2 + floor(6 / 9 + 3 * 2) / 2 - 3

h. 2 + ceil(6 / 9 + 3 * 2) / 2 - 3

5. Create a vector x with the elements ...

a. 2, 4, 6, 8, ...
b. 10, 8, 6, 4, 2, 0, -2, -4
c. 1, 1/2, 1/3, 1/4, 1/5, ...
d. 0, 1/2, 2/3, 3/4, 4/5, ...

The function "rats" just displays the contents of a variable

```
a = 2:2:20  (or whatever max #)
b = 10:-2:-4
c1 = 1:5, c2 = 1./c1 , rats(c2)
d1 = 0:4, d2 = 1:5, d3 = d1./d2 , rats(d3)
```

6. Create a vector x with the elements,

$$x_n = (-1)^{n+1}/(2n-1)$$

Add up the elements of the version of this vector that has 100 elements.

```
n = 1:100;
x = ( (-1).^(n+1) ) ./ (2*n - 1);
y = sum(x)
```

7. Write down the MATLAB expression(s) that will

a. ... compute the length of the hypotenuse of a right triangle given the lengths of the sides (try to do this for a vector of side-length values).

b. ... compute the length of the third side of a triangle given the lengths of the other two sides, given the cosine rule

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

where t is the included angle between the given sides.

8. Given a *vector*, t, of length n, write down the MATLAB expressions that will correctly compute the following:

a. $\ln(2 + t + t^2)$
b. $e^t(1 + \cos(3t))$
c. $\cos^2(t) + \sin^2(t)$
d. $\tan^{-1}(1)$ (this is the *inverse* tangent function)
e. $\cot(t)$
f. $\sec^2(t) + \cot(t) - 1$

Test that your solution works for t = 1:0.2:2

```
t = 1:0.2:2
    a = log(2 + t + t.^2)
    b = exp(t).*(1 + cos(3*t))
    c = cos(t).^2 + sin(t).^2  (all ones!)
    d = atan(t)
    e = cot(t)
    f = sec(t).^2 + cot(t) – 1
```

9. Plot the functions x, x^3, e^x and e^{x2} over the interval 0 < x < 4 ...

a. on rectangular paper
b. on semilog paper (logarithm on the y-axis)
c. on log-log paper

Be sure to use an appropriate mesh of x values to get a smooth set of curves.

10.Make a good plot (i.e., a non-choppy plot) of the function

$f(x) = \sin(1/x)$

For $0.01 < x < 0.1$. How did you create x so that the plot looked good?

11. In polar coordinates (r,t), the equation of an ellipse with one of its foci at the origin is

$$r(t) = a(1 - e^2)/(1 - (e)\cos(t))$$

where a is the size of the semi-major axis (along the x-axis) and e is the eccentricity. Plot ellipses using this formula, ensuring that the curves are smooth by selecting an appropriate number of points in the angular (t) coordinate. Use the command **axis equal** to set the proper axis ratio to see the ellipses.

12. Plot the expression (determined in modeling the growth of the US population)

$$P(t) = 197{,}273{,}000/(1 + e^{-0.0313(t - 1913.25)})$$

where t is the date, in years AD, using t = 1790 to 2000. What population is predicted in the year 2020?

```
t = 1790:2000;
    term = 1 + exp(-0.0313*(t - 1913.25));
    P = 197273000./term;
    plot(t,P)
    xlabel('year'), ylabel('population')
    P2020 = 197273000/(1 + exp(-0.0313*(2020 - 1913.25)))
```

B. Relational/Logical exercises

13. given that x = [1 5 2 8 9 0 1] and y = [5 2 2 6 0 0 2], execute and Explain the results of the following commands:

a. x > y

b. y < x

c. x == y

d. x <= y

e. y >= x

f. x | y

g. x & y

h. x & (~y)

i. (x > y) | (y < x)

j. (x > y) & (y < x)

14. The exercises here show the techniques of logical-indexing (indexing with 0-1 vectors). Given x = 1:10 and y = [3 1 5 6 8 2 9 4 7 0], execute and interpret the results of the following commands:

a. (x > 3) & (x < 8)

b. x(x > 5)

c. y(x <= 4)

d. x((x < 2) | (x >= 8))

e. y((x < 2) | (x >= 8))

f. x(y < 0)

15. The introduction of the **logical** data type in v5.3 has forced some changes in the use of non-logical 0-1 vectors as indices for subscripting. You can see the differences by executing the following commands that attempt to extract the elements of y that correspond to either the odd (a.) or even (b.) elements of x:

a. y(rem(x,2)) vs. y(logical(rem(x,2)))

b. y(~rem(x,2)) vs. y(~logical(rem(x,2)))

16. Given x = [3 15 9 12 -1 0 -12 9 6 1], provide the command(s) that will

a. ... set the values of x that are positive to zero

b. ... set values that are multiples of 3 to 3 (rem will help here)

c. ... multiply the values of x that are even by 5

d. ... extract the values of x that are greater than 10 into a vector called y

e. ... set the values in x that are less than the mean to zero

f. ... set the values in x that are above the mean to their difference from the mean

Ans. x = [3 15 9 12 -1 0 -12 9 6 1]

a = x, idxa = x > 0, a(idxa) = 0

b = x, idxb = ~rem(x,3), b(idxb) = 3

c = x, idxc = ~rem(x,2), c(idxc) = 5*c(idxc)

17. Create the vector x = randperm(35) and then evaluate the following function using only logical indexing:

y(x) = 2 if x < 6

= x - 4 if 6 <= x < 20

= 36 - x if 20 <= x <= 35

You can check your answer by plotting y vs. x with symbols. The curve should be a triangular shape, always above zero and with a maximum of 16. It might also be useful to try setting x to 1:35. Using multiple steps (or a simple Mfile) is recommended for this problem.

Ans;

```
x = 1:35;
  y = zeros(size(x));
  idx1 = x < 6;
  idx2 = (x >= 6) & (x < 20);
  idx3 = (x >= 20) & (x <= 35);
  y(idx1) = 2;
  y(idx2) = x(idx2) - 4;
  y(idx3) = 36 - x(idx3);

  disp([x(:) idx1(:) idx2(:) idx3(:) y(:)])
  plot(x,y,'o')
```

C. IF-block exercises

18.
```
if n > 1          a. n = 7   m = ?
   m = n+1        b. n = 0   m = ?
else              c. n = -10 m = ?
   m = n - 1
end
```

n = 7 gives m = 8

n = 9 gives m = -1

n = -10 gives m = -11

19. if z < 5 a. z = 1 w = ?

```
  w = 2*z          b. z = 9    w = ?
elseif z < 10      c. z = 60   w = ?
  w = 9 - z        d. z = 200  w = ?
elseif z < 100
  w = sqrt(z)
else
  w = z
end
```

z = 1 gives w = 2

z = 9 gives w = 0

z = 60 gives w = sqrt(60)

z = 200 gives w = 200

20.
```
  if T < 30          a. T = 50    h = ?
    h = 2*T + 1      b. T = 15    h = ?
  elseif T < 10      c. T = 0     h = ?
    h = T - 2
  else
    h = 0
  end
```

Ans: T = 50 gives h = 0

T = 15 gives h = 31

T = 0 gives h = 1

21.
```
  if 0 < x < 10          a. x = -1   y = ?
    y = 4*x              b. x = 5    y = ?
  elseif 10 < x < 40     c. x = 30   y = ?
    y = 10*x             d. x = 100  y = ?
  else
    y = 500
```

end

Ans:

x = -1 gives y = -4

x = 5 gives y = 20

x = 30 gives y = 120

x = 100 gives y = 400

In this last exercise, **y = 4x** for any input! Relational operations are subject to precedence and ordering just as arithmetical operations. The *logical* phrase *a* < *x* < *b* should be rendered as the compound logical expression **((a < x) & (x < b))** in MATLAB

Write brief scripts to evaluate the following functions. If you start each script with a request for input (using **input**), you'll be able to test that your code provides the correct results.

22. h(T) = T - 10 when 0 < T < 100

= 0.45 T + 900 when T > 100

Test cases: a. T = 5, h = -5

b. T = 110, h = 949.5

23. f(x) = -1 if x < 0

= 0 if x = 0

= 1 if x > 0

Compare your results to the MATLAB function **sign**.

24. t(y) = 200 when y is below 10,000

= 200 + 0.1 (y - 10,000) when y is between 10,000 and 20,000

= 1,200 + 0.15 (y - 20,000) when y is between 20,000 and 50,000

= 5,700 + 0.25 (y - 50,000) when y is above 50,000

Test cases: a. y = 5,000 t = 200

b. y = 17,000 t = 900

b. y = 25,000 t = 1,950

c. y = 75,000 t = 11,950

25. Explain why the following if-block would **not** be a correct solution to the previous exercise.

```
if y < 10000
   t = 200
elseif 10000 < y < 20000
   t = 200 + 0.1*(y - 10000)
elseif 20000 < y < 50000
   t = 1200 + 0.15*(y - 20000)
elseif y > 50000
   t = 5700 + 0.25*(y - 50000)
end
```

Ans;

See the discussion under IF-block exercise 4, above

C. Array and Matrix exercises

```
a = [1 2 3 4 6 4 3 4 5]          b = a + 2   b = 1×9

plot(b)
grid on
bar(b)
xlabel('Sample #')
ylabel('Pounds')
plot(b,'*')
axis([0 10 0 10])
```

```
A = [1 2 0; 2 5 -1; 4 10 -1]

B = A'

C = A * B

C = A .* B
```

Let's use the matrix A to solve the equation, A*x = b. We do this by using the \ (backslash) operator.

```
b = [1;3;5]

x = A\b

r = A*x - b

eig(A)

svd(A)
```

The "poly" function generates a vector containing the coefficients of the characteristic polynomial.

The characteristic polynomial of a matrix A is

$$det(\lambda I - A)$$

```
p = round(poly(A))

roots(p)

q = conv(p,p)
```

```
r = conv(p,q)

plot(r);
```

At any time, we can get a listing of the variables we have stored in memory using the who or whos command.

```
Whos
```

You can get the value of a particular variable by typing its name.

```
A

sqrt(-1)
```

```
row vector: r = [7 8 9 10 11]

c = [7;  8;  9;  10; 11]

v = [ 1; 2; 3; 4; 5; 6];             % creating a column vector of 6 elements

v(3)
```

When you reference a vector with a colon, such as v(:), all the components of the vector are listed.

```
v = [ 1; 2; 3; 4; 5; 6];             % creating a column vector of 6 elements

v(:

rv = [1 2 3 4 5 6 7 8 9];

sub_rv = rv(3:7)
```

For example, let us create a 4-by-5 matrix *a* −

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8]
```

To reference an element in the m^{th} row and n^{th} column, of a matrix *mx*, we write −

```
mx(m, n)

a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

v = a(:,4)
```

You can also select the elements in the m^{th} through n^{th} columns, for this we write −

```
a(:,m:n)

a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

a(:, 2:3)

a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

a(:, 2:3)

a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

sa = a(2:3,2:4)
```

For example, let us delete the fourth row of a −

```
a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

a( 4 , : ) = []

a = [ 1 2 3 4 5; 2 3 4 5 6; 3 4 5 6 7; 4 5 6 7 8];

a(: , 5)=[]

a = [ 1 2 3 ; 4 5 6; 7 8 9];

new_mat = a([2,3,2,3],:)
```

Multidimensional Arrays

```
a = [7 9 5; 6 1 9; 4 3 2]
```

The array *a* is a 3-by-3 array; we can add a third dimension to *a*, by providing the values like −

```
a(:, :, 2)= [ 1 2 3; 4 5 6; 7 8 9]

B = cat(dim, A1, A2...)
```

Where,

- *B* is the new array created
- *A1*, *A2*, ... are the arrays to be concatenated
- *dim* is the dimension along which to concatenate the arrays

Example

Create a script file and type the following code into it −

```
a = [9 8 7; 6 5 4; 3 2 1];

b = [1 2 3; 4 5 6; 7 8 9];

c = cat(3, a, b, [ 2 3 1; 4 7 8; 3 9 0])
```

26. Given x = [3 1 5 7 9 2 6], explain what the following commands "mean" by by summarizing the net result of the command.

a. x(3)

b. x(1:7)

c. x(1:end)

d. x(1:end-1)

e. x(6:-2:1)

f. x([1 6 2 1 1])

g. sum(x)

27. Given the array A = [2 4 1 ; 6 7 2 ; 3 5 9], provide the commands needed to

a. assign the first row of A to a vector called x1

b. assign the last 2 rows of A to an array called y

c. compute the sum over the columns of A

d. compute the sum over the rows of A

e. compute the standard error of the mean of each column of A (NB. the standard error of the mean is defined as the standard deviation divided by the square root of the number of elements used to compute the mean.)

Ans:

A = [2 4 1 ; 6 7 2 ; 3 5 9]

x1 = A(1,:)

y = A(end-1:end,:)

c = sum(A)

d = sum(A,2) **or** d = sum(A')'

N = size(A,1), e = std(A)/sqrt(N)

28. Given the arrays x = [1 4 8], y = [2 1 5] and A = [3 1 6 ; 5 2 7], determine which of the following statements will correctly execute and provide the result. If the command will not correctly execute, state why it will not. Using the command **whos** may be helpful here.

a. x + y

b. x + A

c. x' + y

d. A - [x' y']

e. [x ; y']

f. [x ; y]

g. A - 3

29. Given the array A = [2 7 9 7; 3 1 5 6 ; 8 1 2 5], explain the results of the Following commands:

a. A'

b. A(:,[1 4])

c. A([2 3],[3 1])

d. reshape(A,2,6)

e. A(:)

f. flipud(A)

g. fliplr(A)

h. [A A(end,:)]

i. A(1:3,:)

j. [A ; A(1:2,:)]

k. sum(A)

l. sum(A')

m. sum(A,2)

k. [[A ; sum(A)] [sum(A,2) ; sum(A(:))]]

30. Given the array A from problem 4, above, provide the command that will

a. assign the even-numbered columns of A to an array called B

b. assign the odd-numbered rows to an array called C

c. convert A into a 4-by-3 array

d. compute the reciprocal of each element of A

e. compute the square-root of each element of A

Ans:

A = [2 7 9 7 ; 3 1 5 6 ; 8 1 2 5]

```
B = A(:,2:2:end)
C = A(1:2:end,:)
```

c = reshape(A,4,3) **or** c = A' (they are different but are both 4x3)

```
d = 1./A , rats(d)
e = sqrt(A)
```

31. Give the following commands to create an array called F:

```
>> randn('seed',123456789)
>> F = randn(5,10);
```

a. Compute the mean of each column and assign the results to the elements of a vector called avg.

b. Compute the standard deviation of each column and assign the results to the elements of a vector called s.

c. Compute the vector of t-scores that test the hypothesis that the mean of each column is no different from zero.

d. If Pr(|t| > 2.132) = 0.1 with 4 degrees of freedom, are any of the mean values in the vector avg statistically different from 0?

Ans:

```
randn('seed',123456789)
F = randn(5,10);
N = size(F,1)
avg = mean(F)
s = std(F)
tscore = (avg - 0)./(s/sqrt(N))
```

None were different at 90% LOC (all < 2.132).

D. Loop constructs:

The answers here provide one version of the solutions. Alternatives are possible and encouraged, especially where time and efficiency of the code is important.

32. Given the vector x = [1 8 3 9 0 1], create a short set of commands that will

a. Add up the values of the elements (Check with **sum**.)

b. Computes the running sum (for element j, the running sum is the sum of the elements from 1 to j, inclusive. Check with **cumsum**.)

c. computes the sine of the given x-values (should be a vector)

Ans:

```
x = [1 8 3 9 0 1]

   a.total = 0;
  for j = 1:length(x)
    total = total + x(j);
  end
b. runningTotal = zeros(size(x));
  runningTotal(1) = x(1);
  for j = 2:length(x)
    runningTotal(j) = runningTotal(j-1) + x(j);
  end
c. s = zeros(size(x));
  for j = 1:length(x)
    s(j) = sin(x(j));
  end
```

33. Create an M-by-N array of random numbers (use **rand**). Move through the array, element by element, and set any value that is less than 0.2 to 0 and any value that is greater than (or equal to) 0.2 to 1.

```
A = rand(4,7);
[M,N] = size(A);
for j = 1:M
  for k = 1:N
    if A(j,k) < 0.2
      A(j,k) = 0;
    else
      A(j,k) = 1;
    end
  end
end
```

34. Given x = [4 1 6] and y = [6 2 7], compute the following arrays

a. $a_{ij} = x_i y_j$

b. $b_{ij} = x_i/y_j$

c. $c_i = x_i y_i$, then add up the elements of c.

d. $d_{ij} = x_i/(2 + x_i + y_j)$

e. e_{ij} = reciprocal of the lesser of x_i and y_j

```
x = [4 1 6], y = [6 2 7]

N = length(x);
for j = 1:N
  c(j) = x(j)*y(j);
  for k = 1:N
    a(j,k) = x(j)*y(k);
    b(j,k) = x(j)/y(k);
    d(j,k) = x(j)/(2 + x(j) + y(k));
```

```
    e(j,k) = 1/min(x(j),y(k));
  end
end
c = sum©;      % or use 1.a. loop
```

35. Write a script that will use the random-number generator **rand** to determine the following:

a. The number of random numbers it takes to add up to 20 (or more).
b. The number of random numbers it takes before a number between 0.8 and 0.85 occurs.
c. The number of random numbers it takes before the mean of those numbers is within 0.01 of 0.5 (the mean of this random-number generator).

It will be worthwhile to run your script several times because you are dealing with random numbers. Can you predict any of the results that are described above?

These code snippets do the job but their repeated use is much more interesting. An example is given for the first exercise.

```
a. total = 0;           % initialize current sum (the test variable)
  count = 0;            % initialize the counter (output of the program)
  while total < 20      % loop until 20 is exceeded
    count = count + 1; % another loop repeat => another number added
    x = rand(1,1);
    total = total + x; % modify the test variable!
  End
  disp([‘It took ‘,int2str(count),’ numbers this time.’])

  ----------------------------------------------------------
```

To do this many times, place the above code in a **for**-loop.
Some simple (though perhaps subtle) changes are needed wth respect
to retaining the counts for each repeat. Also, the summary has
been changed from a single text message to a histogram.

```
Nrep = 1000;   % collect 1000 repeats of the above code
count = zeros(Nrep,1);
for j = 1:Nrep
  total = 0;          % reset the test variable each repeat!!!
  While total < 20
    count(j) = count(j) + 1; % use a vector to capture each result
    total = total +  rand(1,1);
  end
end
hist(count,min(count):max(count))
xlabel('Number of random numbers from U(0,1) added to make 20')
ylabel('Count')
title(['Histogram of results for ',int2str(Nrep),' repeats'])
```

----

b.
```
count = 0;
while 1              % infinite loop use
  count = count + 1;
  x = rand(1,1);                 % get a number
  if (x < 0.85) & (x > 0.8)   % check the value
    break                        % bail out if in selected range
  end
end
disp(['It took ',int2str(count),' numbers this time.'])
```

c.
```
count = 0;
avg = 0;          % test variable
```

```
while abs(avg – 0.5) > 0.01
  count = count + 1;

% The following line is one way to update the average.
% (count-1)*avg is the sum of the first count-1 numbers
% and rand just adds another number.  Dividing by count
% then gives the new average.

  Avg = ((count-1)*avg + rand(1,1))/count;  % modify the test var.

% There are other ways to do this and you are encouraged
% to come up with them

end
disp([‘It took ‘,int2str(count),’ numbers this time.’])
```

Write brief scripts to evaluate the following functions. If you start each script with a request for input (using **input**), you'll be able to test that your code provides the correct results.

36. Write a script that asks for a temperature (in degrees Fahrenheit) and computes the equivalent temperature in degrees Celcius. The script should keep running until no number is provided to convert. [NB. the function **isempty** will be useful here.]

Ans;

```
while 1     % use of an infinite loop
    TinF = input(‘Temperature in F: ‘);  % get input
    if isempty(TinF)                   % how to get out
      break
    end
    TinC = 5*(TinF – 32)/9;            % conversion
```

```
  disp(' ')
  disp([' ➔ Temperature in C = ',num2str(TinC)])
  disp(' ')
end
```