Chapter 8 Relational and Logical Operations

8.1 Relational Operators

Relational and logical operators are instrumental in program flow control. They are used in MATLAB m-Files to test various conditions involving variables and expressions. The relational operators are listed below.

Relational Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
~=	not equal to

Relational operators can be used to compare elements of a numerical array to a scalar. The result is an identically sized array of 1's and 0's indicating the results of the comparison. The resulting array is called a logical array.

Example 8.1.1

```
mu=10;
sigma=1; n=7;
x=mu+sigma*randn(1,n)  % Generate array x of n Normally distributed
                       % numbers with mean mu and std deviation sigma
high1=x>mu  % Find components of x which exceed mu and store in logical
            % array high1

 x =  10.7119   11.2902   10.6686   11.1908    8.7975    9.9802  9.8433
high1= 1         1         1         1         0         0       0

x
low1=x<mu  % Find components of x which are less than mu and store in
           % logical array low1

 x =  10.7119   11.2902   10.6686   11.1908    8.7975    9.9802  9.8433
low1 = 0         0         0         0         1         1       1

x
high2=x>mu+sigma % Find components of x which are greater than mu +
                 % sigma and store in logical array high2

 x =   10.7119  11.2902   10.6686  11.1908   8.7975     9.9802   9.8433
high2=  0        1         0        1        0          0        0

x
low2=x<mu-sigma  % Find components of x which are less than mu -
                 % sigma and store in logical array low2
```

```
 x = 10.7119   11.2902   10.6686   11.1908    8.7975    9.9802   9.8433
low2= 0         0         0         0         1         0        0
```

Example 8.1.2

```
first_name_1='Harold'; last_name_1='Klee';
first_name_2='Robert'; last_name_2='Pratt';
first_name_3='Harold'; lastd_name_3='Smith';
first_name_4='Michael'; last_name_4='Grant';

size_fn_1=size(first_name_1)  % Get size of character string
                              % first_name_1
size_fn_2=size(first_name_2)  % Get size of character string
                              % first_name_2
size_fn_1==size_fn_2 % Check if 1´2 arrays size_fn_1 and
                     % size_fn_2 are equal
match_fn_12=first_name_1==first_name_2  % Check if characters in
                % first_name_1 and characters in first_name_2 are
                % equal and store results in logical array match_fn_12
last_name_2==last_name_4

size_fn_1 =     1     6
size_fn_2 =     1     6
ans =           1     1
match_fn_12 =   0     0     0     0     0     0
ans =           0     1     1     0     1
```

Be careful when checking if two character strings are equal. Remember character strings are actually character string arrays and consequently must be the same size before a comparison using the logical operator '==' can be performed. The following example illustrates the result of trying to compare different size character strings.

Example 8.1.3

```
size(last_name_1)  % Find size of character string last_name_1
size(last_name_2)  % Find size of character string last_name_2
last_name_1==last_name_2 % Invalid operation because last_name_1 and
       % last_name_2 are character string arrays of different sizes

ans =     1     4
ans =     1     5

 ??? Error using ==> ==
Array dimensions must match for binary array op.
```

Example 8.1.4

```
A=[1 5 3; 9 2 5; 6 0 2]
B=eye(3) % Create 3´3 Identity matrix
AB=A==B  % Check locations where A and B are equal and store 1's in
         % those locations of logical array AB and 0's elsewhere
C=2+3*ones(3)
AC=A~=C  % Check locations where A and C are not equal and store 1's in
         % those locations of logical array AC and 0's elsewhere
```

```
A =  1     5     3
     9     2     5
     6     0     2

B =  1     0     0
     0     1     0
     0     0     1

AB = 1     0     0
     0     0     0
     0     1     0

C =  5     5     5
     5     5     5
     5     5     5

AC = 1     0     1
     1     1     0
     1     1     1
```

8.2 Logical Operators

Logical operators are used to negate or combine relational expressions. The truth of falseness of compound expressions comprised of relational operators is possible with the use of logical operators. The standard logical operators are available in MATLAB.

Logical Operator	Description
&	AND
|	OR
~	NOT

Example 8.2.1

```
x=100*rand(1,100);  % Generate an array of 100 random numbers between 0
                    % and 100
y=100*rand(1,100);  % Generate a second array of 100 random numbers
                    % from 0 to 100
xmax=max(x)  % Find maximum number in array x
ymax=max(y)  % Find maximum number in array y
xave=mean(x)  % Find mean of x
yave=mean(y)  % Find mean of y
both=xmax>ymax & xave>yave  % Check if xmax>ymax AND xave>yave are
                            % both true (i.e. logical 1) and set
                            % logical variable 'both' to 1 if so, else
                            % to 0
either=xmax>ymax|xave>yave  % Check if either xmax>ymax OR xave>yave OR
                             % both are true and set logical variable
                             % 'either' to 1 if so, else to 0
~(xmax>ymax)  % If (xmax>ymax) is false, set 'ans' to 1

xmax =  99.9447
ymax =  96.2288
xave =  53.1197
yave =  51.4998
both =   1
either = 1
ans =    0
```

Logical operators make it easy to selectively remove components from an array that fail to satisfy certain criteria. In essence, the original array is multiplied by a similar size logical array of 1's and 0's to produce the final array.

Example 8.2.2

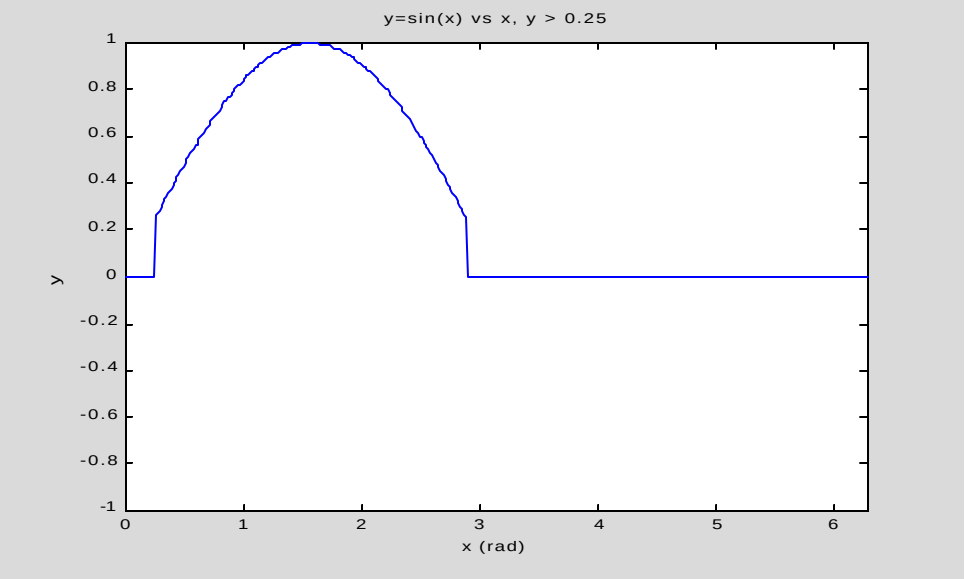
```
n=7;
mu=10; sigma=2;
x=mu+sigma*randn(1,n)  % Generate n Normal deviates with mean mu and
                       % std deviation sigma
x>mu-sigma % Create 1´n logical array 'ans' with 1's wherever components
           % of x satisfy x>mu-sigma and 0's elsewhere
```

```
x=(x>mu-sigma).*x  % Zero out values in array x wherever components of
                   % x do not satisfy x>mu-sigma

x =   6.7918   10.5146    7.8871   12.8303    8.3898   11.0575   10.4386
ans = 0         1         0         1         1         1         1
x =   0        10.5146    0        12.8303    8.3898   11.0575   10.4386

n=7;
mu=10; sigma=2;
x=mu+sigma*randn(1,n)  % Generate n Normal deviates with mean mu and
                       % std deviation sigma
(x>mu-sigma)&(x<mu+sigma)  % Create 1´n logical array 'ans' with 1's
                        % wherever components of x satisfy x>mu-sigma
                        % AND x<mu+sigma and zeros elsewhere
x=((x>mu-sigma)&(x<mu+sigma)).*x  % Zero out values in array x wherever
        % components of x do not satisfy (x>mu-sigma) AND (x<mu+sigma)

x =  8.1562    5.6587    9.8816    7.9787   11.2289   11.0155   13.3849
ans= 1         0         1         0         1         1         0
x =  8.1562    0         9.8816    0        11.2289   11.0155    0
```

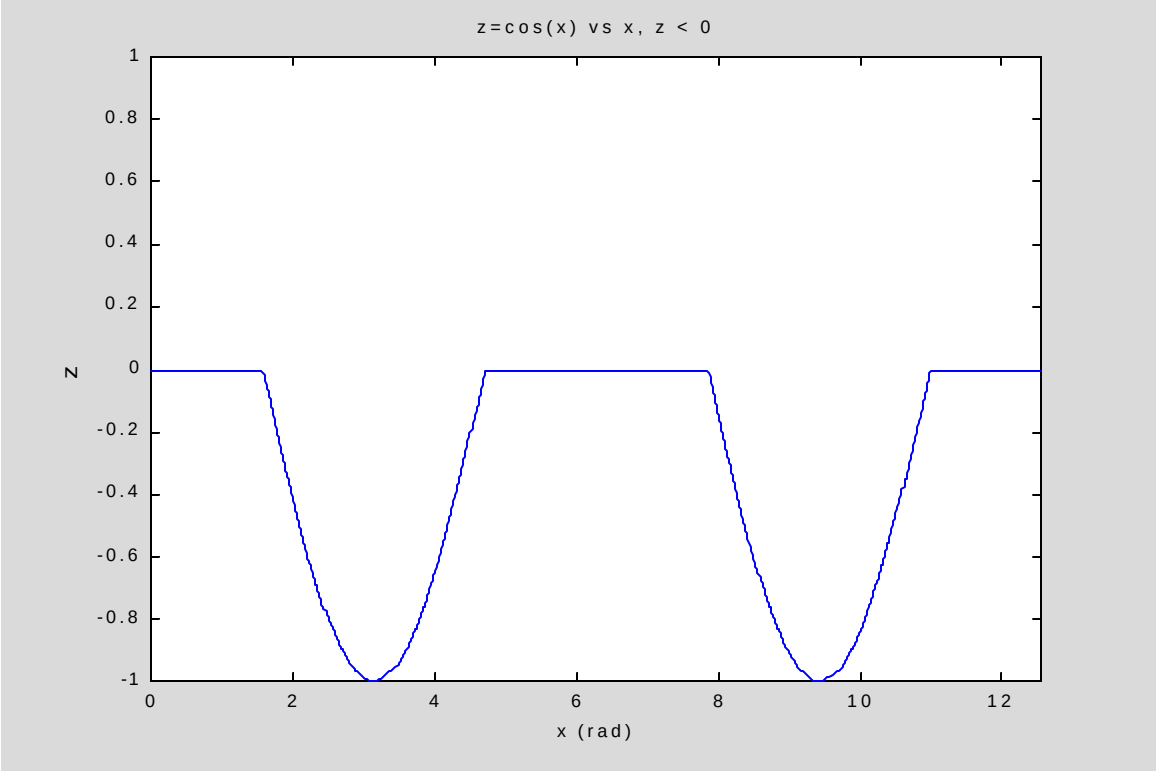
The same concept can be used to plot selected portions of a function over some domain. The following example plots a sine function y = sin(x) whenever y is greater than 0.25 and a cosine function z=cos(x) whenever z is less than zero.

Example 8.2.3

```
x=linspace(0,2*pi,500); % Generate array x of 250 values from 0 to 2*pi
y=sin(x);  % Generate array y of sin(x) values
y=(y>=0.25).*y;  % Zero out values in array y which are less than 0.25
plot(x,y) % Plot modified array y vs array x
axis([0 2*pi -1 1])
xlabel('x (rad)'), ylabel('y'), title('y=sin(x) vs x, y > 0.25')
```



```
x=0:0.01:4*pi;  % Generate array x of values from pi to 4*pi in
                % increments of 0.01
z=cos(x);  % Generate array z of cos(x) values
z=(z<0).*z;  % Zero out values in array z which are greater than 0
plot(x,z)  % plot modified array z vs array x
axis([0 4*pi -1 1])
xlabel('x (rad)'), ylabel('z'), title('z=cos(x) vs x, z < 0')
```



Finally, remember when used with logical operators, MATLAB treats all nonzero numbers as True and only zero to be False.

Example 8.2.4

```
x=3; y=-10; z=0;
a=x&y  % x and y are True, therefore logical variable a is True (1)
b=x&z  % x is True and z is False, therefore logical variable b is
       % False (0)
c=x|y  % x and y are True, therefore logical variable c is True (1)
d=y|z  % y is True and z is False, therefore logical variable d is
       % True (1)
e=~x  % x is True, NOT x is False, therefore e is False (0)
f=~z  % z is False, NOT z is True, therefore f is True (1)

a =     1
b =     0
c =     1
d =     1
e =     0
f =     1
```

8.3 Relational and Logical Functions

MATLAB provides several functions which return logical arrays based on their argument(s) which can be either numerical or character string arrays. These relational and logical functions are:

Function	Description
xor(x,y)	Exclusive OR operation. Return True (1) for each element where either x or y is nonzero (True). Return False(0) where both x and y are zero (False) or both are nonzero.
any(x)	Return True (1) if any element in vector x is nonzero. Return True (1) for each column in a matrix x that has nonzero elements.
all(x)	Rerturn True (1) if all elements in a vector x are nonzero. Return True (1) for each column in a matrix that has all nonzero elements.

Example 8.3.1

```
x=0:10
y=-5:5
xor(x,y)  % Perform exclusive OR on arrays x and y

x =     0     1     2     3     4     5     6     7     8     9    10
y =    -5    -4    -3    -2    -1     0     1     2     3     4     5
ans =   1     0     0     0     0     1     0     0     0     0     0
```

Example 8.3.2

```
A=[1 2 3; 4 5 6; 7 8 9];
B=[1 2 3; 0 5 0; 0 8 9];
C=A-B
any(C)  % Find columns in matrix C which are not all zeros and those
        % which are all zeros

C =  0     0     0
     4     0     6
     7     0     0

ans = 1     0     1
```

Example 8.3.3

```
x=0:0.25:1  % Define set of array x values
y1=x/2  % Create function values for y1
y2=(x-1).^2  % Create function values for y2
ydiff=y1-y2  % Create function values for ydiff
all(ydiff)  % Return 1 if all elements in array 'ydiff' are non-zero
```

```
x =      0        0.2500    0.5000    0.7500    1.0000
y1 =     0        0.1250    0.2500    0.3750    0.5000
y2 =     1.0000   0.5625    0.2500    0.0625    0
ydiff = -1.0000  -0.4375    0         0.3125    0.5000
ans =    0
```

There are other MATLAB functions that return either logical 0 or 1 (or a logical array of 0's and 1's) depending on whether their arguments satisfy certain conditions. Several are tabulated below followed by some examples.

Function	Description
isfinite(x)	True where elements of x are finite
isinf(x)	True where elements of x are infinite
islogical(x)	True if x is a logical array
isnumeric(x)	True if x is a numeric array

Example 8.3.4

```
u=[0 1 1 1 0 1 0] % Create numeric array u
L1=islogical(u)  % Check if u is a logical array by returning 1 in L1
                 % and 0 if its not
L2=isnumeric(u)  % Check if u is a numeric array by returning 1 in L2
                 % and 0 if its not

u =     0     1     1     1     0     1     0
L1 =    0
L2 =    1

x=0:100:1000
format short g
y=exp(x)
L3=isfinite(y)  % Check if elements of array y are finite and place
                % results in logical array L3
L4=isinf(y)  % Check if elements of array y are infinite and place
             % results in logical array L4

x =     0       100         200         300         400         500
      600       700         800         900        1000

y =  1  2.6881e+043   7.226e+086  1.9424e+130  5.2215e+173  1.4036e+217
     3.773e+260       1.0142e+304   Inf          Inf          Inf

L3 =     1     1     1     1     1     1     1     1     0     0     0
L4 =     0     0     0     0     0     0     0     0     1     1     1
```

Function	Description
ischar(x)	True if x is a character string array
isequal(x,y)	True if arrays x and y are identical
isletter(x)	True where elements of array x are letters
isspace(x)	True where elements of x are blanks

Example 8.3.5

```
team=char('Tom', 'Mike', 'Robert', 'Paul', 'Mike', 'Anthony')
     % Create character string array team
size(team)  % Get dimensions of character string array team
ischar(team)  % Check if team is a string array, i.e. all characters
isletter(team)  % Get elements of character array team which are
                % letters
isequal(team(1,:),team(2,:))  % Check if rows 1 and 2 of array team
                              % are the same
isequal(team(2,:),team(5,:))  % Check if rows 2 and 5 of array team
                              % are the same
isspace(team)

team = Tom
       Mike
       Robert
       Paul
       Mike
       Anthony

ans =     6     7
ans =     1

ans = 1     1     1     0     0     0     0
      1     1     1     1     0     0     0
      1     1     1     1     1     1     0
      1     1     1     1     0     0     0
      1     1     1     1     0     0     0
      1     1     1     1     1     1     1

ans = 0
ans = 1

ans = 0     0     0     1     1     1     1
      0     0     0     0     1     1     1
      0     0     0     0     0     0     1
      0     0     0     0     1     1     1
      0     0     0     0     1     1     1
      0     0     0     0     0     0     0
```

8.4 NaNs and Empty Arrays

Certain mathematical operations produce non-numerical results, e.g. 0/0 or ∞/∞. MATLAB recognizes when this occurs and produces a result called NaN, short for 'Not a Number'. When this occurs, subsequent operations on NaN's can generate misleading results. In particular, array comparisons where some elements are NaN's should be avoided because the results are not what one would expect.

Example 8.4.1

```
x1=8; x2=5; x3=2; x4=4; x5=7;
y=(1-((x1-x2)/rem(x1,x2)))/(1-((x2-x3)/rem(x5,x4)))
z=(x1/(x4-2*x3))/(x5/(x1-2*x4))
u=[x1 2*y x2 x3]  % Create array u
w=[2*x4 z+1 x5 x4-x3]  % Create array w
u==w  % Compare elements of arrays u and w

Warning: Divide by zero.
y =   NaN

Warning: Divide by zero.
z =    NaN

u =    8   NaN     5     2
w =    8   NaN     7     2
ans =  1    0      0     1
```

Note that `y` and `z` are NaN's as are expressions `2*y` and `z+1`. However, comparison of arrays `u` and `w` produces a False result for the second element despite the fact that each is a NaN. As far as MATLAB is concerned, two NaN's are not equal. Consequently, MATLAB provides a useful logical function called '`isnan(x)`' which identifies elements of an array `x` which are NaN's.

Example 8.4.2

```
x1=8; x2=5; x3=2; x4=4; x5=7;
x=[x1 x2 x3 x4 x5]
isnan(x)  % Check if elements of array x=[x1 x2 x3 x4 x5] are NaN's
y=(1-((x1-x2)/rem(x1,x2)))/(1-((x2-x3)/rem(x5,x4)))
isnan(y)  % Check if the scalar y is a NaN
u=[x1 2*y x2 x3]  % Create array u
isnan(u)  % Check for NaN's in array u

x =   8     5     2     4     7
ans = 0     0     0     0     0

Warning: Divide by zero.
y =   NaN
ans = 1

u =   8   NaN     5     2
ans = 0     1     0     0
```

The indices of NaN's in an array are easily determined using the '`find`' function.

Example 8.4.3

```
 u=[x1 2*y x2 x3]  % Create array u
w=[2*x4 z+1 x5 x4-x3]  % Create array w
A=[u;w]  % Create array A
A_NaN_indices=find(isnan(A))  % Find indices of NaN elements in A
[i,j]=find(isnan(A))  % Find i,j indices of NaN elements in A
```

```
u =   8   NaN     5     2
w =   8     1     7     2

A = 8   NaN     5     2
    8     1     7     2

A_NaN_indices =  3

i = 1
j = 2
```

There may be ocassions where none of an array's elements satisfy a tested condition. For example, a random sample from a Normal population with mean μ and standard deviation σ will contain values ouside the interval $\mu \pm 3\sigma$ less than 1 % of the time on average. The MATLAB '`find`' function will return an empty array `[]` if the logical array looking for those extreme values contains all zeros.

Example 8.4.4

```
mu=10; sigma=2; n=6;
A=mu+sigma*randn(n);% Generate n´n array of N(m,s) Normal random numbers
L=abs(A)>mu+3*sigma  % Find outliers and store in logical array L
x=find(L)  % Find non-zero indices of elements in array L
size_x=size(x)  % Find size of array containing indices of outliers
```

```
A = 8.9627       9.2511       9.9175     11.313      11.862    12.679
   10.655        7.6282       7.7433      7.6644     10.022    10.579
   10.468        7.8882       7.3014      9.0788      8.7097   12.958
   10.043       12.945        9.4778      9.4751     11.611    12.276
    7.9921      10.111       11.907       7.5737     10.463     8.6317
    8.1057       7.5654      10.257       7.3611      8.0205    7.4161

L =  0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0

x =   []

size_x =  0     0
```

Note, the dimensions returned from the '`size`' function call when the argument is an empty array, i.e. 0 by 0. It's possible to detect the existence of an empty array by checking if its '`length`' is zero; however the more direct approach is to use the logical function '`isempty`'.

Example 8.4.5

```
length_x=length(x)   % Check length of array x
isempty(x)   %  Check if array x is empty

length_x =   0
ans =   1
```