Matlab Index Manipulation

A fair amount of Matlab programming involves the use of matrix or vector indices to select submatrices or subvectors. The simplest is direct specification using the builtin operators, e.g. A(:,1) to select the first column of matrix A. Matlab also has comparsion and boolean operators to select matrix elements based on simple arithmetic tests.

Relational Operators

Matlab has 6 relational or comparison operators

==, ~=	equal to, not equal to
>, >=	greater than, greater than or equal to
<, <=	less than, less than or equal to.

These are inherently "dot" operation and work on an element by element basis. From the matlab help (help relop)

> "A < B does element by element comparisons between A and B and returns a matrix of the same size with elements set to one where the relation is true and elements set to zero where it is not. A and B must have the same dimensions (or one can be a scalar).

```
>> a = [1:2:15]
a =
    1    3    5    7    9   11   13   15
>> ia = (a >= 5)       % The result of the comparison is vector which can be assigned to a name
ia =
    0    0    1    1    1    1    1    1
>> find(a >= 5)       % The builtin function find can be used to get the corresponding indices
ans =
    3    4    5    6    7    8

>> a(a>=5)           % Or you can get the corresponding values directly from a
ans =
    5    7    9   11   13   15


>> A = magic(3)     % 3x3 magic square
A =
    8    1    6
    3    5    7
    4    9    2

>> A >= 5              % works as expected
ans =
    1    0    1
    0    1    1
    0    1    0

>> find(A>=8)       % works on the vector form of A,  A(:) = [ 8 3 4 1 5 9 6 7 3]';
ans =
    1
    6

[ix,jx] = find(A>=8)    % use two return variables to find row, column containing values >= 8.
ix =
    1
    3
jx =
    1
    2
```

```
A(A>=8)     % Also produces a vector result
ans =
   8
   9

A.* (A>=8)       % Use .* and (A >=8) as a mask to preserve matrix shape.
ans =
   8   0   0
   0   0   0
   0   9   0
```

Logical Operators

There are 4 basic logical or boolean operators in matlab that can be used to combine the results of comparison tests (which yield 1's & 0.'s)

&	logical AND
|	logical OR
~	logical complement (NOT)
xor	exclusive OR.

Thus going back to vector a

```
>> a
a =
   1   3   5   7   9   11   13   15

>> (a >= 5) & (a <= 10)
ans =
   0   0   1   1   1   0   0   0

>> a((a >= 5) & (a <= 10))
ans =
   5   7   9

>> find (a < 5 | a > 10)
ans =
   1   2   6   7   8
```

Matlab also has whole suite of built in set functions which can also be used to combine indices found by simpler tests. For example, union and intersect are functions that produce results equivalent to the operators & and | plus find. Type "help union" or "help intersect" for further details

```
>> ia = find(a<5)
ia =
   1   2
>> ib = find(a > 10)
ib =
   6   7   8
>> union(ia,ib)     % Combine the two sets of indices, i.e do an OR.
ans =
   1   2   6   7   8
```

example: plot a subsection of the Mt Soledad map near Scripps

```
load ljtopo1       % ljh, ljgeo
ljlon = linspace(ljgeo.x_min,ljgeo.x_max,ljgeo.nx);
ljlat  = linspace(ljgeo.y_min,ljgeo.y_max,ljgeo.ny);

ixlat = find(ljlat >= 32.852 & ljlat <= 32.872);
ixlon = find(ljlon >= -117.26 & ljlon <= -117.23);
s = surf(ljlon(ixlon),ljlat(ixlat),ljh(ixlat,ixlon));
```

Related functions

There are two functions any and all which which return true (1) if any or all elements of a vector pass some test

```
a =
    1    3    5    7    9   11   13   15
>> any(a > 13)
ans =
    1
>> all(a > 13)
ans =
    0
```

For matrices these functions, like a lot of matlab functions, work on a column by column basis by default, e.g sum, prod

```
A =
    8    1    6
    3    5    7
    4    9    2

>> any(A>8)        % By default any works on a matrix column by column
ans =
    0    1    0

>> any(A(:) > 8)     % Use the vector form of A to get a single scalar answer
ans =
    1
```

You are unlikely to use either function directly but they are often used implicitly in the if statement to get a scalar result

```
>> if (a >5)           % really if all(a>5)
  disp('yes')        % if test is true
else
  disp('no')         % if test is false
end

no

>> if (A  < 10 )           % really if all(A(:)<10)
  disp('yes')        % if test is true
else
  disp('no')         % if test is false
end

yes
```

Sort function

The sort function as its name suggests sorts a vector into ascending order, but it also will return the indices in the order need to perform the sort. Simple example :-

```
>> a = [3, 2, 5, 8, 1, 7, 6, 4,9];
>> [asorted,ixsort] = sort(a)       % produces a sorted vector + index vector to sort the original vector
    1    2    3    4    5    6    7    8    9
ixsort =
    5    2    1    8    3    7    6    4    9
>> a(ixsort)
ans =
    1    2    3    4    5    6    7    8    9
```

Practical example, sorting rows of the gravity data matrix for plotting

```
data = load('class_grav05.asc');
  ht = data(35:48,4);             % Access the 2002 & 2003 data
```

```
lon = data(35:48,3);
plot(lon,ht)
[slon,ixs] = sort(lon);      % get sorted longitude and index vector for the sort

plot(lon(ixs),ht(ixs))    % simultaneously sort/permute lon & ht for plotting
hold on
plot(lon,ht,'*r')
```

Unique function

The unique function as its name suggests find the unique elements of a vector, and also an index vector to select the unique elements. NOTE: unique will also sort the elements of the vector, if this is not what you want use the sort function on the returned index vector.

Example:

```
>> c = [-1 3, 2, 5, -1, 8, 1, 7, -1, 6, 4,9, -1];
>> [cuniq,ixuniq] = unique(c)
cuniq =
   -1    1    2    3    4    5    6    7    8    9  % sorted c with repeated -1's removed
ixuniq =
   13    7    3    2   11    4   10    8    6   12
>> c(ixuniq)                                  % using the index vector
ans =
   -1    1    2    3    4    5    6    7    8    9

>> ixalt = sort(ixuniq);   % if you want to preseve the original order but with repetitions removed
>> c(ixalt)
ans =
    3    2    5    8    1    7    6    4    9   -1
```

Practical usage: To remove the repeated base station measurements from the gravity dataset

Miscellaneous

For loop

A "for loop" will cause a block of code between the initial for statement and the corresponding closing end to be executed repeatedly with different values of a control variable. In the following example, ix is the control variable and it is set to successive elements of the vector [1:5]

```
for ix = [1:5]
 fprintf(1,'ix is now ');
 fprintf(1,'%d ',ix);
 fprintf(1,'\n');
  pause(1)     % Tell matlab to wait for 1 second so we can see each iteration
end

ix is now 1
ix is now 2
ix is now 3
ix is now 4
ix is now 5
```

NOTE: In the for statement the assignment is really to successive columns from a matrix. Thus we can do the following

```
A = [8    1    6    % Magic square with all row/columns adding to 15 created using A = magic(3);
     3    5    7
     4    9    2];
for ix = A
  ....  % Same code as above
end

ix is now 8 3 4
```

```
ix is now 1 5 9
ix is now 6 7 2
```

A consequence of this rule is that a frequent cause of errors is the accident use a column vector, rather than a row vector. In the first example with a column vector, the loop would execute only once with ix assigned to the entire vector.

```
for ix = [1:5]'
 ...    % Same code as above
end

ix is now 1 2 3 4 5
```

NOTE: Because Matlab can do many calculations in parallel using vector notation the "for loop" is used much less frequently than its equivalent in other languages such as Fortran or C. When a calculation can be done using vector notation rather than a for loop it will be much faster, perhaps 100x faster, and more elegant.