Other MATLAB Functions

Leigh J. Little SUNY Brockport October 12, 2022

Contents

1 Introduction

We have started to investigate vectorized calculations in MATLAB. When MATLAB was originally written, the use of such operations was critical. This is because MATLAB experienced a huge slowdown in its execution speed when it had to perform looping operations. Now days this situation has improved, but using vectorized operations in MATLAB is still recommended when feasible. This can makes your code easier to interpret and write, however if many different forms of vectorization, array indicies and vector subscripts are used in a single operation, it can make understand what a command is doing challenging. Being able to work with vectorized operations is critical to understanding advanced MATLAB code.

2 Special Subscripts

We have seen that subscripts have to be positive integers or vectors of positive integers, but there are two special exceptions.

2.1 The `end` Subscript

It is easy to get the first element of a vector; this is just `x(1)`. Getting the last element of a vector can also be done, but it is not as convenient. You can always get the last element of a vector using `x(length(x))`. However, an alternative is to use `x(end)`.

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> x(end)
ans =
     3
```

2.2 The `:` Subscript

Another special subscript is the `:` subscript. Type in the following:

```
>> x = [4 -2 0 3]
x =
     4  -2   0   3
>> x(:)
ans =
     4
    -2
     0
     3
```

Notice that this has an effect similar to the transpose function (or apostrophe operator). What if `x` is a column vector?

```
>> x = [5 3 1 -1]'
x =
    5
    3
    1
   -1
>> x(:)
ans =
     5
     3
     1
    -1
```

In this case, the subscript has no effect. If the vector is a row vector, then the `:` subscript transposes the vector. If the vector is a column vector it has no effect.

3 The `sum` Function

The `sum` function can be used to sum the elements of a vector

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> sum(x)
ans =
      7
```

You can also apply other functions to `x` prior to summing. For example

```
>> sum(abs(x))
ans =
      29
>>sum(cos(x))
ans =
    -9.819784871666869e-01
```

You can also sum only a portion of `x` using an array index (or other vector subscript)

```
>> sum(x(1:2:end))
ans =
     4
```

The above statement will sum the odd subscripted elements of `x`.

4 The `prod` Function

Similar to the sum function, the `prod` function will multiply the elements of a vector together

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> prod(x)
ans =
      -10800
```

This function provides an easy way to compute $n!$. Suppose we wanted to compute 5!. Then we could do

```
>> prod(1:5)
ans =
   120
```

5 The `max` Function

As its name implies, the `max` command will return the maximum element of a vector.

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> max(x)
ans =
     5
```

This function provides a good opportunity to demonstrate a key feature of MATLAB. Many MATLAB functions can return more than one output. For example, type in the following:

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> [xmax, ii] = max(x)
xmax =
     5
ii =
     2
>> x(ii)
ans =
      5
```

In this case, we have asked MATLAB to compute the maximum value of `x` and store this in `xmax`. We have also asked for the first location that this maximum value occurs in and to store this in the variable `ii`.

In the event that we only want the location of the maximum and not the maximum value itself, we can do

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> [~, ii] = max(x)
ii =
     2
>> x(ii)
ans =
      5
```

We need to put the tilde character in the first position of the expression on the left side of the `=` sign as a place holder. There is also a `min` command that behaves the same way.

6 The `sort` Function

The `sort` command will sort a vector in increasing order, though this behavior can be changed.

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> sort(x)
ans =
    -6    -3    -2     1     3     4     5     5
```

It we want to sort the vector in decreasing order, we can do one of two things:

```
>> -sort(-x)
ans =
     5     5     4     3     1    -2    -3    -6
>> sort(x,'descend')
ans =
     5     5     4     3     1    -2    -3    -6
```

As with the `max` function, the `sort` function can return multiple outputs. Suppose you entered

```
>> [xs, ii] = sort(x)
xs =
   -6    -3    -2     1     3     4     5     5
ii =
    3     4     6     1     8     7     2     5
```

Here we have stored the sorted vector in the variable `xs`, but what is the meaning of the `ii` vector? The `ii` vector indicates how the vector was sorted. For example,

```
ii(1) = 3   =>   xs(1) was originally x(3)
ii(2) = 4   =>   xs(2) was originally x(4)
ii(3) = 6   =>   xs(3) was originally x(6)
```

7 The `find` Function

The `find` function is one of the most useful functions in MATLAB. Its purpose is to locate elements of a vector that satisfy a desired criteria. Almost any search that you would want to do can be performed using the `find` command. For example if we want to find all of the elements of a vector that are positive, we can do

```
>> x = [1 5 -6 -3 5 -2 4 3]
x =
     1     5    -6    -3     5    -2     4     3
>> ii = find(x > 0)
ii =
    1     2     5     7     8
>> x(ii)
ans =
     1     5     5     4     3
```

The output of a `find` command is always going to be a set of subscripts (or an empty vector if none of the elements matches the given criteria).

```
>> x = [1 5 -6 -3 5 -2 4 3]
>> ii = find(x > 10)
ii =
  1x0 empty double row vector
```

Here the `find` command is returning an empty vector because there are no elements of `x` that are greater than 10.

If we wanted to find all elements of `x` that are even we could do

```
>> x = [1 5 -6 -3 5 -2 4 3]
>> ii = find(mod(x,2) == 0)
ii =
    3     6     7
>> x(ii)
ans =
    -6    -2     4
```

Here, `ii` is the set of subscripts where `x` is even. The `x(ii)` are the actual even elements of `x`.

What about a more challenging example? Suppose `x` was a vector of angles in radians and we wanted to find all angles that are in Quadrant 2. Then we could do

```
ii = find(x > pi/2 & x < pi)
```

An interesting fact; the x- and y-axes are not in any quadrant, so the inequalities above should not include the `=` sign. The `find` command is mathematically correct, but recall that rounding errors in the values of `x` could result in some elements being incorrectly excluded (or included). This is a situation where the problem lies not in the `find` command, but in the calculation of the elements of `x`.

8 State Functions

MATLAB has a series of functions called *state functions*. These functions are essentially specific instances of an `if` test. These are also called the `is` functions. There are about 50 of these functions. To get a list of these, you can open the main MATLAB help window and search for *state functions* or `is*`. The output of these functions is always true or false (1 or 0). Some common examples of these are given below.

```
isempty(x)          -> checks if x is an empty vector
isinf(x)            -> checks if x is infinity
isnan(x)            -> checks if x is not a number
isvector(x)         -> checks if x is a vector
ismember(x,set)     -> checks if x is a member of a given set.
                       ismember(1,[4 5 6]) = 0 because 1 is not
                       in the set 4, 5, 6
isa(x,type)         -> checks if x is of the given type.  This
                       can be used to check if x is an integer,
                       double, character, etc.
```