An Introduction to Matlab: Part 3

This lecture assumes that you have already worked through part 1 and part 2. You should be able to use many basic Matlab commands and use Matlab as a calculator on scalar variables. You should be able to create and execute a script file. This lecture covers

- Creating row/column vectors
- Vector operations
- Applying functions to vectors
- Component wise arithmetic

Creating vectors

This section goes through creating row and column vectors by typing each element, by using patterns, and by using a few built in functions.

1. *Typing in vectors explicitly:* Here we learn how to type vectors.
   (a) Open Matlab. If you already have it open, type *clear all;* in the *Command Window*. You may do everything here in a script file or on the *Command Window*. I would suggest simply using the *Command Window* for now so you don't have to rerun you file everytime we make a change.
   (b) Let us create a row vector consisting of the elements $1, 3, 5$ and 7. To do this, we use the bracket notation []. We assign this vector to the variable u.
   u=[1,3,5,7]
   Instead of using commas, try simply inserting a space between each element. Type
   u=[1 3 5 7]
   Note that they are the same thing. Also note that this is a row vector (Matlab prints out the numbers in a row).
   (c) Let us create a column vector consisting of the same elements $1, 3, 5$ and 7. For a row vector, we put either spaces or commas in between elements. For a column vector, we put a semicolon between each element. Note that a semicolon is also used to supress output, but Matlab is smart enough to know that a semicolon within brackets [] means to change rows. Type
   v=[1;3;5;7]
   Note that Matlab displays the elements of v as a column.
2. *Typing in vectors using a pattern:* We create vectors which are large and correspond to a distinct pattern.
   (a) Suppose we want to create a row vector, u, that contains elements 1 through 100 (ie $u = [1, 2, \ldots, 99, 100]$). How can we do this? Well, you can type in each element, but that seems awfully silly. Matlab has a nice way to do this. Type
   u=[1:1:100]
   This notation means *u goes from 1 by 1 to 100.* Create the vector $u = [1, 3, 5, 7, 9, 11]$ using this notation. How do you think you would create the vector $u = [15, 12, 9, 6, 3, 0]$? Well, it goes from 15 by -3 to 0. So we type
   u=[15:-3:0]
   We can also use negative numbers. Create a vector that goes from -15 to 0 by 3.
   (b) Note that using the notation above (*u=[15:-3:0]*) created a row vector. Let us say we want a column vector instead. The easiest way to do this is to use the *transpose* operator, which is the single quote: '. Type:
   u=[15:-3:0]
   v=u'
   What happened? Well, u is created as a row vector, then v is the transpose of u, which is a column vector. We could have simply done this on one line also by typing
   v = [15:-3:0]'
   Try to create a column vector that goes from 2 to 16 by 2.

3. *Creating a zero vector, one vector, or random vector:* Here we discuss how to create a vector of all zeros, all ones, or of completely random entries (between 0 and 1)

   (a) To create a vector of all zeros, we use the *zeros* function. It takes two (for now) arguments. These represent the size of the vector you wish to create. Type:
   u = zeros(5,1)
   You get a column vector with 5 zeros. Try to create a row vector with 10 entries.

   (b) To create a vector of all ones, we use the *ones* function. It has the same structure as the zeros function:
   v = ones(1,6)

   (c) You can also create a vector with random entries (these random entries are randomly drawn from between 0 and 1) using the *rand* function:
   u = rand(4,1)

Vector operations

This section goes through basic vector arithmetic and how to create new vectors using parts of old vectors.

1. *Vector addition and scalar multiplication:* We learn how to add vectors and scale vectors.

   (a) Type *clear all;* Create three row vectors, u, v and w by doing the following:
   u = [1 3 5 7]
   v = [0 1 2 4]
   w = [2 4 6 8 10]
   Add vectors u and v by typing:
   u+v
   Note that your result is another vector. We could assign it a varible by typing *r=u+v* .
   Now add v and w. What happens? We get an error because they are not the same size. Recall that we can only add vectors of the same size.
   Create a new vector, *v1*, that is the transpose of v , again by typing *:*
   v1=v';
   Try adding *v1* and v. What happens? An error again. Why? Because row vectors and column vectors cannot be added together.

   (b) How do you suppose we multiply a vector by a scalar? Well, exactly how you would expect! Let us multiply w by *2* by typing:
   *2*w*
   Find πu and $e^2 v$ by doing the same.

2. *New vectors from old vectors:* Here we'll learn how to incorporate an old vector as part of a new vector, or how to extract part of an old vector.

   (a) We'll use the same u, v, and w from above, so recreate the vectors if need be. Let us say that I only want to look at the 2^{nd} element (component) of vector v. How might I do this? Well, vectors are indexed from 1 until their size (4 in this case) and are accessed using parenthesis. So typing:
   v(2)
   returns the 2^{nd} element of v. Use this idea to find the 1^{st} and 4^{th} components of v.

   (b) Let us say that I want more than one element of v. Say I want to get a new vector that is the same as v, but without the 2^{nd} element. So I want a new vector, call it $v1$, that is size 3 an contains the 1^{st}, 3^{rd} and 4^{th} components of v. I could type:
   v1=[v(1) v(3) v(4)]
   but for very large vectors, this will not be feasible. Another way I could do this is by typing:

v1=v([1 3 4])

which says take only the parts with index in *[1 3 4]*. This is still a bit difficult for large vectors. Luckily, we can use the colon notation (like when we defined *u=[15:-3:0])*. So I can instead type:

v1=v([1 3:4])

Create a new vector r that goes from 3 by 3 to 99. From this vector, create another vector *r1* that contains the first 3 elements of r, the 15^{th} element, the 18^{th} element and the last 4 elements (elements 30 through 33).

(c) We could also take elements out of order. For example, using the r defined previously, lets say we wanted a new vector *r1* that was the last element (33^{rd}), the 10^{th} element, then elements $5, 4, 3$ and 2. You would do this exactly as expected:

v1=v([33 10 5:-1:2]).

(d) Now, let us consider the problem of enlarging a vector, rather than taking part of a previous vector. Let us create the vector $t = [0, 1, 2, 4, 6, 8]$ by taking $v = [0, 1, 2, 4]$ and adding 6 and 8 to the end. There are two ways to do this:

i. Simply define $t = v$ then $t(5) = 6$ and $t(6) = 8$. This takes previously undefined value of t and defines them, thus enlarging t. Try this.

ii. Define a new vector that is exactly what you want, v, with a 6 and 8 tacked on:
t=[v,6,8]

Use this idea to create a new vector $d = [-2, 0, 1, 2, 4, 10]$ by adding -2 to the beginning of v and 10 to the end. You can also combine two vectors, by doing something like:

s=[v,t]

(e) Note that we have used row vectors, but all of the above operations hold for column vectors as well. The difference is that notation such as

s=[v,t]

will not make sense if v and t are arbitrary column vectors. Instead we'd want to use

s=[v ; t]

to make sure the rows from t appear after the rows from v.

Applying functions to vectors

As most of you have seen in your classes, there are many useful functions that apply to vectors. For example, one might want to know the norm of a vector or the dimension of a vector. In this section, we use a handful of these functions. These are by no means all the useful functions, but just a small subset.

1. *Functions on a single vector:* Here we discuss functions that take a single vector as the argument, or apply to single vectors.

(a) Type *clear all;* and define $a = [1, 0, 1]$, $b = [0, 0, 1]$, $c = [1 : 1 : 10]$ and $d = [2 : 2 : 20]$.

(b) Recall the norm of a vector $x = [x_1, \ldots, x_n]$ is defined as

$$\|x\| = \sqrt{x_1^2 + \cdots + x_n^2}.$$

The Matlab function for the norm is, oddly enough, *norm.* To find the norm of c, type:

norm(c)

Using norm, we can create a unit vector simply by dividing a given vector by its norm. Create a vector v that is a unit vector in the same direction as d.

(c) To find the length or size (dimension) of a vector, we can use the functions *length* or *size*. *length* returns one argument, while *size* returns two. Type:

[n m] = size(c)

k = length(c)

Notice the difference between the two functions.

2. *Functions on multiple vectors:* Here we discuss functions that take multiple vectors as arguments

   (a) We will be using the same vectors as previously, so redefine a, b, c and d if necessary.

   (b) Let us start with the dot product. Recall for two vectors of the same size, $c \cdot d$ is the sum of the product of each individual component. As you may guess, the dot product function is *dot*. Try:

   dot(c,d)

   What about *dot(a,c)?* What happens?

   (c) If two vectors are in $\mathbb{R}^3$, then recall that we can define the cross product, which gives a third vector perpendicular to both vectors. Try:

   cross(a,b)

   What happens if you try *cross(c,d)?*

3. There are numerous other functions. We'll just list some here. Try them out and type *help FunctionName* for more help

   (a) Find minimum/maximum: *min(c), max(c)*

   (b) Calculate absolute value: *abs([-1 0 1])*

   (c) Calculate the sum of the entries: *sum(a)*

   (d) Calculate the product of the entries: *prod(c)*

Component wise arithmetic

We can also manipulate vectors using ***component wise arithmetic.*** The reason this is in bold print and gets its own section is because this is a very hard concept for beginning Matlab users, but one that is essential to fully understand Matlab. Let us say that I want to square the entries of the vector d. How can I do this? Well, try:

d^2

What happens? It doesn't work, why not? Because I cannot multiply d times itself, because that type of vector multiplication makes no sense. Matlab has a way to take care of this. We add a period . before the operation. Try:

d.^2

What happened? Matlab applied the operation *^2* to each part of d. That's what the . meant.

Using the same idea, we can multiply each entry of d by the corresponding entry of c using

*d.*c*

Again, this means multiply *each component* of d to the *corresponding component* of c. Note that this only makes sense if c and d are the same size.

Create a new vector that contains the entries of c divided by the entries of d.