

# Matlab Tutorial

## 1 Generating Vectors

- To generate a row vector, leave a space between each entry. To generate a column vector, put a semicolon after each column:

```
>> A=[1 3 5 7 9 11 13 15]
>> B=[1;3;5;7]
```

- To call up a certain element of a vector, use subscripts:

```
>> A(3)
>> B(5)
```

You should get that the third element of the vector  $A$  is 5, and the following error message for  $B(5)$ :  
??? Index exceeds matrix dimensions.

- To access sequential elements use the colon notation:

```
>> A(1:3)
>> A(5:end)
>> A(3:-1:1)
>> A(2:2:7)
```

The following four commands do different things.  $A(1:3)$  means “start at the first entry and display each element to the third one,”  $A(5:end)$  means “start at the fifth element and display each one until the end,” and  $A(3:-1:1)$  means “start at the third element, display every element counting down by one each time to the first element.” Can you figure out what the fourth element displays?

- For a vector, you can access the size by the command `length`:

```
>> length(A)
```

- To delete an entry, use the following command:

```
>> A(3)=[];
```

- The following is a list to make specific vectors:

```
>> x=first:last           % Create a row vector starting with ‘‘first’’,
                          % counting by 1, ending at or before ‘‘last’’

>> x=first:increment:last % Create row vector starting with first, counting
                          % by increment and ending at first or before last

>> x=linspace(first,last,n) % Create linearly spaced row vector starting with
                          % first, ending at last, having n elements

>> x=logspace(first,last,n) % Create logarithmically (base 10) spaced row vector
                          % starting with 10^first, ending at 10^last, having
                          % n elements
```

## 2 Generating Matrices

- To generate a matrix, you use a space to separate elements in a specific row, and semicolons are used to separate individual rows.

```
>> A=[1 3 4 5 7; 5 6 10 9 11; 10 23 23 1 3]
```

This will generate a matrix with 3 rows and 5 columns

- Indexing matrix entries follows standard convention. `>> A(3,3)` calls up the entry in the third row and the third column of *A*. The colon is used in a similar manner to display entire rows or columns:

```
>> A(:,5)           % Displays all the rows and the fifth column of A
>> A(1,:)           % Displays the first row and all the columns of A
>> A(2,3:4)         % Displays row 2 and columns 3 and 4 of A
>> A(1:2,2:5)       % Displays rows 1 and 2 and columns 2 through 5 of A
```

To assign other values to a particular row or column, you must assign things of equal sizes:

```
>> A(:,3) = [3 5 7] % Will not work because you are assigning a row vector to
                % a column vector
```

```
>> A(:,3) = [3;5;7] % This should do the trick.
```

- You can't delete individual matrix entries (I don't know how to do multidimensional arrays), but if you wish to delete a particular row or column, you can use a similar command like we did for vectors:

```
>> A(:,3)=[]       % Deletes the third column from A. The matrix is now 3 by 4
```

- To determine the size of your matrix, use the `size` command, which returns a 2 element row vector, the first entry being your row dimension, the second your column dimension:

```
>> k=size(A)       % Assigns k as a 2 element row vector
```

- The following is a list of matrices you can generate (there are additional options for all of these, and you can access them by typing `help` command at the command prompt:

```
>> A=zeros(M,N)    % Creates an M by N matrix of zeros
```

```
>> A=ones(M,N)     % Creates an M by N matrix of ones
```

```
>> A=rand(M,N)     % Creates an M by N matrix of random entries normally distributed
                % between 0 and 1
```

```
>> A=randn(M,N)    % Creates an M by N matrix of random entries normally distributed
                % between -1 and 1
```

## 3 Matrix & Vector Mathematics

- To add, subtract, or multiply a scalar multiple to a matrix or vector, you do the following:

```
>> A-2             % Subtract 2 from every element (addition works similarly)
>> 2*A            % Multiply each element by 2
```

**Division does not follow the same!** This is a key point—to divide every element by a scalar, you must precede the division sign (/) with a period (.).

```
>> B=A./2      % Divide every element by 2
```

- If you have two arrays of the same size, and you want to multiply each element of the two arrays, then you precede the multiplication (\*) by period (.). Exponentiation of arrays is the same:

```
>> A.*B      % Multiply corresponding elements in A and B
```

- Matrix multiplication is defined by the multiplication sign as long as you have two matrices that can be multiplied ( $AB$  is defined when the columns of  $A$  have the same dimension as the rows of  $B$ ).
- Solving a linear system  $Ax = b$  is quick and easy to do. Recall that for a unique solution, you need to have as many equations as you do unknowns ( $A$  is a square matrix. You first need to check that your system will have a unique solution, easily done by using the rank command. (Recall a fact from linear algebra: if you have  $n$  equations and  $n$  unknowns, then if your rank is  $n$ , you have a unique solution).

```
>> A=rand(100,100);    % Generate a 100 by 100 random matrix
>> b=randn(100,1);    % Generate a 100 by 1 random matrix
>> rank(A)             % Test the rank
>> x=b\A;             % Solve the system
```

Note that we have solved the system  $x = A^{-1}b$ . In Matlab, you can find the inverse of a matrix  $A$  with the command `inv(A)`, but for solving linear systems it is easier (and faster) to use the commands above.

- To find specific matrix entries, it is quite easy to do. Matrices are also indexed sequentially as numbers (i.e.  $A(4)$  is the same as  $A(1,4)$ ), but I tend to avoid this as it may be confusing if you forget the size of your matrix. However, knowing this can be quite handy if you want to set specific entries to a value:

```
>> A=round(100*rand(100,100));    % Create a random matrix, multiply each entry
                                   % by 10, and then round up each entry.

>> indices=find(A>80);            % Find all indices bigger than 80.
>> A(indices)=0;                  % Set all of those indices to 0.
```

- To write for loops, it is never ever advisable to write a doubly scripted for loop (Matlab becomes extremely slow!) It is better to write one for loop, vectorizing your code. Let's work a similar example as before, using Matlab's clock functions to illustrate this.

### Example 1

```
>> A=round(100*rand(1000,1000));
>> tic;                               % Starts Matlab's clock
>> [row col]=size(A);
>> for i=1:row
>> for j=1:col
>> if(A(i,j)>80) A(i,j)=0;
>> end
>> end
>> end
>> toc                               % Ends Matlab's clock
```

## Example 2

```
>> A=round(100*rand(1000,1000));
>> tic;
>> [row col]=size(A);
>> for i=1:row
>> indices=find(A(i,:)>80);
>> A(i,indices)=0;
>> end
>> toc
```

## 4 Writing Executable Programs

One way to use Matlab is through the command line, however if you write a lot of statements, it becomes clumsy (and time consuming) to re-enter in the material over and over again. A way around this is to make function M files.

Function M files are easy to create and generate. For the purposes of this class we won't get too fancy, although you will need to know the basics.

Open up a text editing program for use. I like to use `nedit`, and get to it by typing `nedit &` in a terminal window. Remember the `&` allows you to still execute commands in the terminal. If you use the GUI version of Matlab it has a text editor as well.

If you are using `nedit`, there are two helpful things you should do to get started. Select the "Preferences" button with your mouse and select "Show Line Numbers." This should put lines of code on the left hand side of your screen. If you have errors in your M-files, Matlab will generally refer to the specific line number where the error occurs, so this is helpful. Also, under the "Preferences" button, select "Language Mode." This should make another pull down menu to the right. Select "Matlab." Now the program recognizes special characters and commands specific to Matlab.

On the first line in the text editor, you want to type `function program_name`. The word "function" calls specific attention to the fact that it is an M-file. "program\_name" is the name of your program you wish to save. It might be a good idea to save your program now. (Ctrl-S). When you do this, (a) make sure to give the file name the same name as `program_name` in your text editor, and (b) then add a ".m" at the end, so you should type "program\_name.m" You are all set now to do things. Any lines you type Matlab will execute as commands as if you were entering them in on the screen.

On the course website there is a file called "simple\_program.m". Save this file to your current working directory and open it in `nedit`. At the Matlab prompt, all you need to do is type `simple_program` and it will execute the statements. This program will execute a few basic statements and should work you through plotting.

## 5 Resources

1. Duane Hanselman & Bruce Littlefield, *Mastering Matlab 7*. Prentice Hall (2004).