

# A Gentle Introduction to Matlab

Alexey Koloydenko

November 24, 2009

## 1 What is *Matlab*?

MATLAB is a powerful environment for scientific computing, modelling, and software development. MATLAB is a commercial (and rather expensive) software package developed by *The Mathworks* <http://www.mathworks.com> .

## 2 Starting and exiting *Matlab*

To **start** MATLAB, double-click on the MATLAB shortcut on your desktop and wait for your session to initialise. Eventually, a matlab desktop will appear. In the desktop, look for the *Command Window*, it has “>>” as its command line prompt.

You can exit by MATLAB typing either `exit` or `quit` in the command line, or following the menu **File** > **Exit Matlab**.

## 3 Have a go with basic expressions

*MATLAB can be immediately used as a scientific calculator:*

```
>> factorial(20)
```

```
ans =
```

```
2.4329e+18
```

```
>> sin(pi/3)+2*sqrt(5)
```

```
ans =
```

```
5.3382
```

```
>> log(exp(0.4))
```

```
ans =
```

```
0.4000
```

```
>> log2(2^6)
```

```
ans =
```

```
6
```

```
>> gcd(3,6)
```

```
ans =
```

```
3
```

```
>> lcm(3,6)
```

```
ans =
```

```
6
```

MATLAB also has its own *programming language* and can interpret your instructions on the fly as well as run longer programs. Those familiar with C or C++

programming will notice some similarities in formatting:

```
>> fprintf('20!=%18.0f\n', factorial(20))
20!=2432902008176640000
```

(This gives the same result as `factorial(20)` above in an unabbreviated format.)

## 4 Getting help

Although many of the MATLAB basic commands have intuitive names

```
>> sqrt(2)
```

```
ans =
```

```
1.4142
```

we still often need to refer to the help pages. Typing `help` or `doc` leads to the top of the hierarchy of the help resources. Getting help on a specific function or topic is also simple. For example, type `help eye` to see how identity matrices can be produced. Copy the given example `x = eye(2,3,'int8')` and paste it into the command line. Note that the role of the closing `;` is to prevent display of the results. Note also that MATLAB has various data types (classes), e.g. 8 bit (1 byte) integer.

MATLAB provides many examples organised in *demos*. Type `demo` and navigate your mouse to the left pane of the newly opened window. Expand the *Mathematics* subtree and go over the *Basic Matrix Operations*.

There are also many free tutorials and webinars provided by the Mathworks as well as third party guides similar to this one.

## 5 Simple Plotting

MATLAB can draw graphs in one

```
>> x=0:0.01:3;
>> plot(x,exp(-x).*x.^2.*sin(x))
```

two and three dimensions:

```
>> [X,Y] = meshgrid(-3:0.2:3);
>> Z=1/3-(2*X.^2+Y.^2)/19;
>> mesh(X,Y,Z)
```

With your mouse click on ‘Rotate 3D’ item (9th in 2nd menu row). Place the mouse pointer on the surface and give it a twirl.

```
>> z=peaks(25);
>> surf1(z);
>> shading interp;
>> figure;
>> colormap(pink);
>> contour(z,16);
>> colormap(hsv);
```

## 6 Modularity

Over the years, MATLAB has grown enormously. In addition to the main MATLAB engine and interpreting environment, there are now many specialised toolboxes on sale from the Mathworks and third parties, as well as numerous free contributions.

To see what is available in your installation, type `ver`.

### 6.1 Symbolic computations

A few years ago, the Mathworks acquired the MAPLE kernel/engine which has resulted in the MATLAB *Symbolic Math Toolbox* supporting symbolic computation and variable-precision arithmetic. The `vpa` command used below stands for *variable precision arithmetic*.

```
>> vpa('pi',50)
```

```
ans =
```

```
3.1415926535897932384626433832795028841971693993751
```

```
>> syms x
```

```
>> factor(6*x^2+18*x-24)
```

```
ans =
```

```
6*(x+4)*(x-1)
```

```
>> pretty(factor(collect(1/(x+1)+2/(x+3))))
```

$$\frac{3x + 5}{(x + 1)(x + 3)}$$

```
>> solve('x^3-6*x^2+11*x-6=0')
```

```
ans =
```

```
1
```

```
2
```

```
3
```

```
>> diff('x^3*cos(x)')
```

```
ans =
```

```
3*x^2*cos(x)-x^3*sin(x)
```

```
>> int('x^2/(x^4-1)')
```

```
ans =
```

```
1/4*log(x-1)-1/4*log(x+1)+1/2*atan(x)
```

```
http://www.mathworks.com/products/symbolic/
```

## 7 Importing Data

Data can be entered into a MATLAB session in several ways, including directly from the low level devices (ports).

Using your internet browser, locate

```
http://personal.rhul.ac.uk/utah/113/acnseed.txt
```

and save this file in your current directory.<sup>1</sup>

Now, go to **File** > **Import Data** and choose the newly saved file. MATLAB is ready to import three variables: a  $22 \times 2$  matrix of numerical data and two variables containing literal labels for each of the columns. Choose **Next** and suppose we are only interested in the numerical part. We then unselect the rest and choose **Finish** to import the data into the workspace.

Data can also be readily input from various multimedia channels.

### 7.1 Loading an image

For example, the following command loads an image file directly from the web:

```
A=imread('http://personal.rhul.ac.uk/utah/113/Picture%20001.jpg');
```

---

<sup>1</sup>To determine your current directory, type the following (UNIX-style) command: `pwd`.

MATLAB treats all data, including images, as arrays.

## 7.2 Exploring image data

Type

```
size(A)
```

to determine the dimensions of the image array. Note that matrices  $\mathbf{A}(:,:,1)$ ,  $\mathbf{A}(:,:,2)$ , and  $\mathbf{A}(:,:,3)$  contain the values of the RED, GREEN, and BLUE colours of the image, respectively. Such images are called *true colour* images. We can actually visualise a single colour. For example, type

```
imshow(A(:,:,1));
```

to see the intensity map of the RED colour component.

Next, use the interactive image tool to explore the image in more detail. Type

```
imtool;
```

and select the image  $\mathbf{A}$  in the drop-down **File**  $\rangle$  **Import from Workspace**. Take a note of the updated “Pixel info” displayed in the left lower corner of the “Image Tool 1 - A” figure: the updates are being made in response to your mouse movements over the image.

The first two numbers are the (*column,row*) coordinates, and the triples in the brackets are the Red, Green, Blue intensities on the 0 – 255 scale.

We next convert the colour image into a grayscale one:

```
>> Agray=rgb2gray(A);
```

```
>> [r, c]=size(Agray)
```

We continue to use this image to illustrate some simple statistical functions of MATLAB.

## 7.3 Distribution of intensities. Image histograms

The image  $\mathbf{Agray}$  is simply a matrix with integer entries in the range of 0 (dark) to 255 (bright). This range is in some sense a legacy of the old displays that could map

only so few distinct values on the screen.

Disregarding the geometric information, the image **Agray** can simply be thought of a set of  $r \times c$  intensity values. Hence it makes sense to compute various statistics of this set. Since some of the forthcoming computations will not respect the original *uint8* data type, we convert *Agray* to the type *double*:

```
Adouble=double(Agray);
```

Now,

```
>> m=min(Adouble(:))
>> M=max(Adouble(:))
>> mn=mean(Adouble(:))
>> s=std(Adouble(:))
>> md=median(Adouble(:))
>> qts=prctile(Adouble(:),[5,25,50,75,95])
```

The last command computes 5th, 25th,...percentiles of the image intensity data. Note also how the  $r \times c$  array *Adouble* was implicitly reshaped into  $rc \times 1$  (column) vector. If we did not reshape (and convert) the image, then we would need to use specialised versions of the otherwise general statistical functions. For example, in order to get a more detailed information about the intensity distribution of the image, we use

```
>> figure
>> imhist(Agray)
```

(compare to `hist(Adouble(:))`).

## 7.4 Simulations

MATLAB offers many ways to simulate various processes, including random ones (**Statistics toolbox**). Let us generate an array of uniformly and identically distributed pseudo-random numbers in the same range as the above image

```
>> R=unidrnd(M-m+1, [r, c])-1+m;
>> R=uint8(R);
>> figure;
>> imshow(R);
>> colormap gray(256);
```

and explore a histogram of the resulting image:

```
>> figure
>> imhist(R)
```

## 7.5 Sampling. More synthetic images

Let us make the last example a bit more meaningful. Namely, let us synthesise a new image, say,  $A_2$  that will have (almost) the same histogram as image  $A_{gray}$ , but otherwise  $A_2$  will be completely “random”. In detail, let  $A_2(1, 1), A_2(1, 2), \dots, A_2(1, c), A_2(2, 1), A_2(2, 2), \dots, A_2(2, c), \dots, A_2(r-1, 1), A_2(r-1, 2), \dots, A_2(r-1, c), A_2(r, 1), A_2(r, 2), \dots, A_2(r, c)$  be a sample of *independent identically distributed* random variables following the multinomial distribution identical to the intensity distribution (histogram) of  $A_{gray}$ .

```
>> A2=randsample(Agray(:), r*c, true);
>> A2=reshape(A2, r, c);
```

The first command produces a random sample with replacements (*true*) from the population determined by  $A_{gray}(:)$ . The second command “matrises” vector  $A_2$  by placing its first  $r$  entries into the first column, next  $r$  entries into the second column, and so on.

```
>> figure; imshow(A2); colormap gray(256);
>> figure; imhist(A2);
```

## 8 Deleting and Saving Data

First, let us close all the graphics of this session by typing

```
close all
```

The following command deletes the arrays **A** and **R** from the workspace:

```
clear A R
```

We can actually store the new image **Agray** as a JPEG image that can be viewed by external applications. To do so, type `imwrite(Agray,'grayphoto.jpg')`

Before exiting MATLAB (especially in a hurry:), it might be a good idea to save all your data (if the disk space permits):

```
save mymatlab.mat
```

This saves all your variables in the file named **mymatlab.mat**. When you restart MATLAB, you can load all the data from this file

```
load mymatlab.mat
```

## 9 A bit of programming

People with some programming experience will find programming in MATLAB easy.

Suppose we want to analyse distribution of differences  $Adouble(i+1,j) - Adouble(i,j)$  for all  $i = 1, 2, \dots, r-1$  and  $j = 1, 2, \dots, c$ .

```
>> d=[];  
>> for i=1:r-1,  
for j=1:c,  
d=[d,Adouble(i+1,j)-Adouble(i,j)];  
end;  
end;
```

All  $(r-1) \times c$  differences are now stored in vector  $d$ . (Try `hist(d,100)` to see a histogram of  $d$ .)

Note that the above code took a noticeable time to run. There are two very important points to make here, *memory preallocation* and *vectorisation*.

## 9.1 Memory preallocation

The size of your output ( $d$  in this example) can often be determined in advance. It would then save MATLAB (and yourself) much time if you give MATLAB this information:

```
>> d=zeros(r-1,c);
>> for i=1:r-1,
for j=1:c,
d(i,j)=Adouble(i+1,j)-Adouble(i,j);
end;
end;
>> d=d(:);
```

## 9.2 Vectorisation

Equally important is the fact that nearly all MATLAB functions operate on matrices (as opposed to scalars). This very often allows us to avoid explicit loops (**for**, **while**). Since explicit loops can only partially be optimised by the MATLAB interpreter, avoiding them (“vectorising”) usually gives big savings in runtime.

```
>> d=Adouble(2:end,:)-Adouble(1:end-1,:);
>> d=d(:);
```

Indeed, if you think the operation you want to perform is not too uncommon, chances are MATLAB has a dedicated function for it, which would generally be very efficient:

```
>> d=diff(Adouble);
>> d=d(:);
```

**A further example** (designed by Roland Bunschoten of University of Amsterdam): Given two large sets, say,  $a$  and  $b$ , of points in high dimensional Euclidean space, compute the (Euclidean) distances between each pair of points with one point taken from  $a$  and the other from  $b$ .

```
>> a = rand(400,100); b = rand(400,200);
```

Thus, we put 100 400-dimensional vectors in  $a$  and 200 400-dimensional vectors in  $b$ .

There are several ways of solving this problem in MATLAB without the use of explicit loops. However, the following solution turns out to be the most efficient in both runtime and memory.

The blueprint of the solution is the formula

$$\|A - B\| = \sqrt{\|A\|^2 + \|B\|^2 - 2A \cdot B}$$

which is valid when  $A$  and  $B$  are vectors.

```
>> aa=sum(a.*a); bb=sum(b.*b); ab=a'*b;
>> d=sqrt(repmat(aa',[1 size(bb,2)])+repmat(bb,[size(aa,2) 1])-2*ab);
```

Note that  $d$  is indeed the  $100 \times 200$  matrix of distances. Note also the use of the replication operator *repmat* which brings the vectors of (squared) norms to the common size/dimension. Apparently, this inflation of memory is still well compensated by the efficiency of the vectorised functions.

## 10 Further

Take a look at the examples and demos provided by the Mathworks. For example, <http://www.mathworks.com/access/helpdesk/help/techdoc/math/f4-982991.html> points to the Linear Algebra tools of MATLAB.

Also, consider familiarising yourself with how MATLAB programs can be organised in *functions* and *scripts* [http://www.mathworks.com/access/helpdesk/help/techdoc/learn\\_matlab/f4-2525.html](http://www.mathworks.com/access/helpdesk/help/techdoc/learn_matlab/f4-2525.html)

Another useful set of examples are about the *Optimization Toolbox* and can be found following this link <http://www.mathworks.com/products/optimization/demos.html?file=/products/demos/shipping/optim/tutdemo.html>