Matlab basics

Preliminary step : A good practice is to create a specific folder to save all files corresponding to a Matlab session, and select it as the current directory within Matlab. This can be done using the "Browse to folder" small icon (which should look like this : ).

Arrays and matrices

Matlab was originally developed for matrix manipulation. There are several ways for defining matrices. The first one is to explicitely enter the coefficients : for example the command

```
M = [ 2 1 4 ; 3 2 1 ]
```

defines a 2×3 matrix.

Here are some examples of usual operations and manipulations with matrices (execute each of them to see their effects)

```
M(2,1)
2*M
M+1
N=M
N(1,3)=0
M+N
M-N
M'
[M,N;N,M]
```

Multiplication of matrices is performed using the $*$ operator. Dimensions of matrices must be compatible :

```
M*N
M*N'
```

The / and \operations are used to solve linear systems. See the effects of the following commands :

```
A = [ 2 1 ; 3 2 ]
b = [ 1 ; -1 ]
A\b   % finds X such that AX=b
inv(A)
A^(-1)
inv(A)*b
b/A   % finds X such that XA=b (impossible in this case because of dimensions)
b'/A
b'*inv(A)
```

To perform element-wise multiplications, divisions and powers, one needs an additional dot symbol before the operator :

```
M.*N
M./M
M./N
M.*N'
M.^2
```

Many mathematical functions are available and they are usually "vectorized", which means that they can take matrices as inputs, applying to each element of the matrix :

```
cos(M)
sqrt(M)
```

The function `zeros` creates a matrix of zeros of a specified size :

```
zeros(1,3)
```

The : symbol can be used to create a row vector containing a sequence of consecutive integers. More generally one can use `linspace` to produce regular sequences of values :

```
2:10
linspace(1,2,11)
```

Plotting functions and curves in Matlab

The basic plotting command in Matlab is `plot`. If `x` and `y` are vectors of the same length, then `plot(x,y)` will draw segments joining all points $(x(i), y(i))$

```
x = [1 4 2];
y = [2 0 1];
plot(x,y)
```

Hence to draw a function $y = f(x)$ one needs to specify a vector of values for x and the corresponding $y = f(x)$ values. For example :

```
x = linspace(-pi,pi,500);
y = cos(x);
plot(x,y)
```

The same command can be used to draw parametrized curves $\gamma(t) = (x(t), y(t))$. For example, to draw a circle :

```
t = linspace(-pi,pi,500);
x = cos(t);
y = sin(t);
figure % opens a new figure
plot(x,y)
```

Functions of two variables can be plotted as surfaces using the commands `surf` or `mesh`. For example, suppose we want to plot the function $f(x, y) = xy$ over the range of values $-1 \leq x \leq 1$, $-1 \leq y \leq 1$. We first need to create a grid of points in the square $[-1, 1]^2$. This can be done directly in Matlab using the command `meshgrid` :

```
x = linspace(-1,1,30);
y = linspace(-1,1,30);
[X,Y] = meshgrid(x,y);
```

We compute the value of the function on every point in the grid :

```
Z = X.*Y;
```

Finally we plot the surface :

```
surf(X,Y,Z)
```

User-defined functions

We can create our own functions using the "@" symbol. For example, the following lines define a function named `cosplusone` which computes $\cos(x)+1$ from input x, and then use this function twice :

```
cosplusone = @(x)cos(x)+1; %
cosplusone(pi/3)
cosplusone(M)
```

User-defined functions can also be defined using the editor and saving them into files. For example to create the function `cosplustwo` which will compute $\cos(x)+2$, click on the yellow "+" button on top. This opens a new file in the editor. Then type in

```
function y = cosplustwo(x)
 % computes cos(x)+2
 y = cos(x)+2;
end
```

and save to file named `cosplustwo.m`. You can now use this function at the prompt, exactly as the previous one :

```
cosplustwo(pi/3)
```

Image manipulation

Matlab can load and manipulate images. A gray-level image is represented by a matrix giving the intensity of each pixel. Download the test image hill and save it to your current folder. Then type :

```
I = imread('hill.png');
I(1:10,1:10)
imagesc(I)
colormap gray
```

In this case the values are of integer type, between 0 (black) and 256 (white). In order to perform some computations, we will convert them to real numbers (of double precision type, which is the default in Matlab) :

```
I = double(I);
```

We can edit the image by simply changing the values in the matrix

```
Imod = I;
Imod(100:400,65:115) = 50;
imagesc(Imod)
```

The variations of pixel intensities along the rows and columns can be computed using the function `gradient` :

```
[Gr,Gc] = gradient(I);
imagesc(Gr)
imagesc(Gc)
```

We can compute and display the magnitude of the gradient, i.e. the norm of the gradient vector at each pixel :

```
Gm = sqrt(Gr.^2+Gc.^2);
imagesc(Gm)
```

We see that the gradient magnitude is large along the edges of the image. Thus we can think of selecting edges by thresholding its values :

```
imagesc(Gm>=10)
imagesc(Gm>=20)
imagesc(Gm>=30)
```

We can think about defining a function from these operations. Let us create a new file, and type in :

```
function Ie = DetectEdges(I,alpha)
  % Simple edge detector : Ie = DetectEdges(I,alpha) computes image Ie from image I
  % by thresholding the magnitude of its gradient at the level alpha
  [Gr,Gc] = gradient(I);
  Gm = sqrt(Gr.^2+Gc.^2);
  Ie = Gm>=alpha;
end
```

then save it to `DetectEdges.m`.

Scripts

One can write code in a file so that it can be saved and executed all at once. Such files are called Matlab scripts. For example let us save some code from the previous section (image manipulation), so that we can use it later. Create a new file and type in :

```
% script from Matlab tutorial on image manipulation.
clear % delete all variables
close all % close all figures
I = imread('hill.png');
I = double(I)
```

```
subplot(1,2,1) % this is used to draw several figures in the same window.
imagesc(I)
colormap gray
title('Original image')
Ie = DetectEdges(I,20);
subplot(1,2,2)
imagesc(Ie)
colormap gray
title('Simple edge detector')
```

then save it to `DemoEdgeDetect.m`. This code can now be used by typing `DemoEdgeDetect` in the command window (or pressing the "Run" green arrow).

Loops and conditional statements

The following lines show examples of uses of loops and conditional statements. Write these lines in a separate script and execute it :

```
% script from Matlab tutorial : examples of loops and conditional statements

%% example of "for" loop
x = 3;
for i=1:6
    x = x^2
end

%% example with "while" loop and "if" statement
% Find the root of the polynomial x^3 - 2x - 5 using interval bisection.
a = 0;
b = 3;
while b-a > 1e-6
    x = (a+b)/2;
    fx = x^3-2*x-5;
    if fx > 0
        b = x;
    else
        a = x;
    end
end
disp('value found : ')
disp(x)
% show result as a plot
clf % clear current figure
t = linspace(-2,3,1000);
plot(t,t.^3-2*t-5)
hold on % means we want to add more plots in the same figure
plot(x,fx,'*r')
```