CALCULUS - PRACTICAL I - INTRODUCTION

PEDRO FORTUNY AYUSO

This practical gives a simple introduction to Matlab.

1. Basic computations

Matlab works mainly with *vectors*, not with numbers. Another way to looking at it is saying that it works with *lists* (in a sense, a vector is no more than a list). This is the natural and usual way to use it.

1.1. Making lists.

There are two main commands for building lists:

- `linspace`: Generates a list of evenly spaced values between two numbers: `linspace(a, b, n)` generates a list (vector) of n numbers, evenly spaced between a and b. Try it for several values of a, b and n.
- `[a:s:b]`: This instruction generates a list of numbers starting at a, going up to (at most) b and in steps of length s. Try it.

For example, to assign to a variable x the vector of 200 evenly spaced values between $-\pi$ and π, one writes:

```
> x = linspace(-pi, pi, 200);
```

(For what is the semicolon?).

On the other hand, to assign to y the vector of values between $-\pi$ and π in steps of 0.01, one writes:

```
> y = -pi:.01:pi;
```

(For what is the semicolon?).

We are not going to insist on these two instructions, they will be used continually along the course.

1.2. Elementary operations.

Let us assign to x the vector of 300 evenly distributed components between $-\pi$ and π:

```
> x = linspace(-pi, pi, 300);
```

and compute, for example, the sine of each of those values:

```
> y = sin(x);
```

(notice that the semicolon hides the output). Write y without the semicolon to see its value:

```
> y
```

(a list of 300 numbers, each the sine of the corresponding value of x).

Let us plot y against x (notice that y is a vector and x is another one of the same length: we are plotting a *table*):

```
> plot(x,y);
```

A window with the graph of the sine function should appear. Notice that simply using x and y, which are vectors, we have been able to plot the whole family of pairs (x_1, y_1), (x_2, y_2), ..., (x_{300}, y_{300}). This is what makes Matlab so powerful.

Date: 9 de octubre de 2013.

Drill:

(1) Plot the graph of the function $exp(x)$ for x from -5 to 5, using steps of size 0.03.
(2) Plot the graph of the function $x^2 - 3x + 1$, for x from -10 to 10 using 300 points. What happens if one writes `x^2`? An error. Why?
(3) Plot the graph of $tan(x)cos(x)$ for x from -2 to 3 using steps of length 0.025. What happens if one writes `tan(x)*cos(x)`? An error. Why?

1.3. **The dotted (`.`) operators.**

As the student has noticed in the previous examples, if `x` is a vector *one cannot compute* `x*x` (which would be the product of a vector by itself), nor `x^2` (raising to the power 2); an error happens. This is because element-wise operations (which were the desired ones) require a `.` (dot).

In order to multiply, divide and rise to a power vectors elementwise one has to insert a `.` *before* the operator.

```
> a = linspace(-3, 3, 200); b = linspace(-4, 4, 200);
> b = a * b ;% error
> y = a .* b;% correct a×b
> u = a / b;% error
> u = a ./ b;% correct
> v = a ^ 2;% error
> v = a .^ 2;% correct
```

This must be clear. We (in Calculus) shall use the dot *always*. Never forget it. This is unlike in Algebra, where most of the multiplications are matricial, which are written without the dot.

2. Anonymous functions

The second most powerful property of Matlab is the possibility of defining functions easily. Assume that, for whatever reason, we need to use the polynomial $P(x) = 321x^2 - 3x + 1$, say for plotting it several times for different sets of points. Having to write it time and again would be boring and error-prone. To avoid this, one can define a function representing it, an *anonymous function*:

```
> P = @(x) 321.*x.^2 - 3.*x + 1
```

From now on, `P` represents the function $P(x)$, the polynomial written above. It can be used quite easily:

- Plot P between -5 and 5 using 300 points.
- Plot P between $-\pi$ and 20 with steps of 0,025.
- Compute $P(3), P(0), P(2132)$.

Anonymous functions can be of several variables. For example, $f(x, y) = xy - x^2 + y^3$ (*notice the dots!*):

```
> f = @(x,y) x.*y - x.^2 + y.^3;
```

Computing $f(1, 2), f(2, 0), f(-\pi, 7)$ is now trivial.

3. Symbolic calculus

In Calculus, one needs to perform symbolic computations many times (limits, derivatives, integrals. . .). Matlab can handle these but it is necessary to specify the symbolic variables in advance.

Before proceeding, let us clear all the values of the variables:

```
> clear all;
```

(Now neither `x` nor `y` nor any of the variables defined above have any value).

Let us compute the limit of $\left(1+\frac{1}{n}\right)^n$ for n tending to infinite (what is it?). Let us try it straightaway:

```
> limit((1+1./n).^n,n,inf)
```

(`inf` is the symbol for ∞ in Matlab). This gives an error because Matlab *does not know what* `n` *is.* We want it to be a symbol, not a value, so we have to specify it:

```
> syms n
```

And let us now repeat the instruction above:

```
> limit((1+1./n).^n,n,inf)
```

The answer is `exp(1)`, which is what is called e, obviously (or not?).

But there are many more operations that can be performed.

Compute the following:

$$\lim_{x\to 0}\frac{\sin(x)}{x};\quad \lim_{x\to\pi}\frac{(x-\pi)^2}{\tan(x)^2};\quad \lim_{x\to 1}(x-1)\log(1-x)$$
$$\lim_{x\to-\pi}(\cos(x)-1)\tan(x+\pi);\quad \lim_{x\to 0}x^x;\quad \lim_{x\to 3}\frac{x+1}{x-3}.$$

Derivatives of functions are also easily computed:

```
> P =@(t) cos(t).*sin(t) - t.^3 + exp(t).*t
```

defines an anonymous function. If we want to compute its derivative, we need a symbolic variable first (say `u`):

```
> syms u
```

and now we just need to compute the derivative:

```
> diff(P(u),u)
```

the second derivative:

```
> diff(P(u),u,2)
```

the third one:

```
> diff(P(u),u,3)
```

etc.

3.1. From symbolic to anonymous.

In older (which are the ones we are using) versions of Matlab, there is no way to evaluate a symbolic function at a point. For example, let `Q` be the symbolic derivative of `P` above:

```
> syms u; Q = diff(P(u),u)
```

and let us try to evaluate it at 3:

```
> Q(3)
```

gives an error. In order to obtain the equivalent *anonymous* function, the method `matlabFunction` is used. Thus,

```
> Q_n = matlabFunction(Q)
```

creates the function `Q_n`, which is the *anonymous* equivalent of the symbolic one `Q`. This anonymous version can be evaluated:

```
> Q_n(3)
```

gives its value at 3.

4. Exercises

Exercise 1. Given

$$f(x) = \tan(x)x^3 - \cos(x) + \frac{x+1}{x^2+1},$$

do the following:

(1) Define it as an anonymous function.
(2) Plot its graph between -4 and 4 using 300 points.
(3) Plot its graph between -10 and 10 in steps of 0.01.
(4) Compute its limits for x tending to $-\infty$, $+\infty$, 0 and 1.
(5) Compute its first, second and fourth derivatives.

Exercise 2. Same for

$$\begin{aligned} g(x) &= \frac{\tan(x)}{\sin(x/2)} \\ h(u) &= (e^u - 1)\log(|u| + ,01) \\ k(u) &= u^3 - u^7 \\ l(v) &= \frac{v}{v^2+1} \\ m(t) &= \frac{1}{t^4+1} \end{aligned}$$

Exercise 3. Plot, *on the same graph*, the following functions (choosing freely their domains, etc.):

$$f(x) = \sin(x)\cos(x), \quad g(x) = \exp(x)/10.$$

Same (on a different graph) for:

$$f(u) = u^2 - 2u + 1, \quad g(t) = \exp(-t)/10.$$

In order to perform these tasks, one uses the instruction `hold on`, which tells Matlab not to use a new graph for the next plots. The alternative `hold off` tells Matlab to use a new graph for each plot. In order to clear the graphical space, one uses `clf`.

Exercise 4. Plot each of the four functions of the previous exercise in a single figure, all figures inside the same window. Use the `subplot` command for doing this. Use a different domain for each function.