1. Matlab

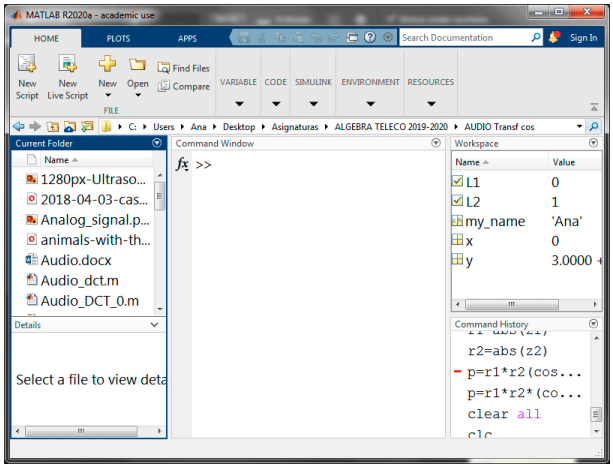
MATLAB (matrix laboratory) is a programming language developed by MathWorks. It started out as a matrix programming language where linear algebra programming was simple.

It allows matrix manipulations; plotting of functions and data; implementation of algorithms; creation of user interfaces; interfacing with programs written in other languages, including C, C++, Java, and Fortran; analyze data; develop algorithms; and create models and applications.

It has numerous built-in commands and math functions that help you in mathematical calculations, generating plots and performing numerical methods.

MATLAB is used in every facet of computational mathematics, and it is commonly used in mathematical computations such as dealing with matrices and arrays, Linear Algebra, Calculus and Differential Equations, Statistics, 2D and 3D plotting and graphics, data analysis, etc.

Lets take a look at Matlab environment!



1.1. Variables

To create a variable it is enough to chose a name, and assign a value to it:

```
>> x=0
>> y=-1/3
>> my_name='Ana'
>> L1=isequal(x,-3)
>> L2=isequal(x,0)
```

A valid name starts with a letter, which can be followed by letters, digits, or underscores. Matlab is case sensitive, so X and x are not the same variable. The maximum length of a variable name is the value that the namelengthmax command returns (in my 2020 version is 63, check your).

Now lets see the variables that we have created and their types with the order `who` or `whos`.

We can observe that the basic type of data in Matlab is the matrix, even the simple numerical variables as `x=0` are internally saved as a 1×1 matrix.

1.2. Creating vectors

Vectors are rows or columns matrices. We create a vector by writing its elements between brackets, separated by blanks, commas or semicolon (semicolon produces column vectors).

```
>> v=[1 3^2 pi 1/3]
>> v=[1,3^2,pi,1/3]
>> w=[1;3^2;pi;1/3]
>> u1=[1:100]
>> u2=[1:4:100]
```

1.3. Creating Matlab files

We will work with two types of Matlab files: scrips and funtions. Both scripts and functions allow you to reuse sequences of commands by storing them in program files. Scripts are the simplest type of program, since they store commands exactly as you would type them at the command line. However, functions are more flexible and more easily extensible.

Ejemplo 1 *Create a script named `circle_area1.m` which computes the area of a circle of radius* $r = 5$.

```
r=25;
a=pi*r^2;
disp('the area is:')
disp(a)
```

An other way of giving the radius *r* is using the `input` statement:

Ejemplo 2 *Create a script named `circle_area2.m` which computes the area of a circle. The program asks you to input the value of the radius r.*

```
r=input('Enter the radius of the circle ');
a=pi*r^2;
disp('the area is:')
disp(a)
```

We can do the same by creating a function:

Ejemplo 3 *Create a function named `circle_area.m` which computes the area of a circle of radius r. The input argument is the radius r, and the output argument is the area a.*

```
function [a] = circle_area(r)
% computes the area of a circle of radius r
% a = the area of the circle of radius r
a=pi*r^2;% the formula
disp('the area is:')
disp(a)
end
```

Which is the main difference between functions and scripts?

1.4. Matlab Control Structures *if*

We can check a condition with the control structure ***if***.

if evaluates an expression, and executes a group of statements when the expression is true.

The `elseif` and `else` blocks are optional and their statements execute only if previous expressions in the if structure are false.

```
if condition1
    statement 1
    statement 2
    ...
elseif condition2
    statements
    ...
else
    statements
    ...
end
```

Ejemplo 4 *Modify the previous function* `circle_area.m` *by adding an if structure verifying that the input radius is not a negative number.*

```
if r>=0
    a=pi*r^2;% the formula
    disp('the area is:')
    disp(a)
else
    error('the radius is a negative number')
end
```

Ejercicio 1 *Create a Matlab function named* `second_degree.m` *that computes the roots of a second degree equation. Consider the coefficients a, b, and c of the equation as inputs. The output will be the vector* `sol=[sol1;sol2]`.

2. Complex Numbers

2.1. Basic Fuctions

The imaginary unit is represented in MATLAB by the letters i or j. This is because MATLAB is used widely in both mathematics (where i is most commonly used for the square root of -1) and (electrical) Engineering (where j is more commonly used).

```
>> i^2
ans =
-1
>> j^2
ans =
-1
>> sqrt(-1)
ans =
0.0000 + 1.0000i
```

Function	Output
`isreal(z)`	0 if `z` has an imaginary part 1 otherwise
`real(z)`	Real part of `z`
`imag(z)`	Imaginary part of `z`
`conj(z)`	Conjugate of `z`
`abs(z)`	Modulus of `z`
`angle(z)`	Argument in $[-\pi,\pi]$
`roots(p)`	Roots of the polynomial p. Input p is a vector containing $n+1$ polynomial coefficients, starting with the coefficient of x^n.

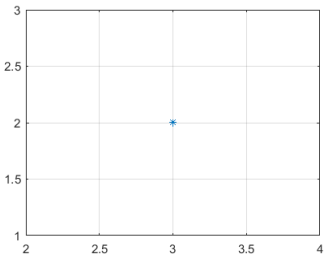
Let $z = 2+3i$. Find the real part, modulus, polar form (with the angle measured in degrees) and complex conjugate of z. Obtain the principal argument of z from its real and imaginary parts.

2.2. Graphical representation

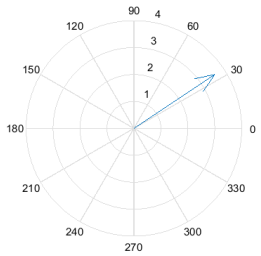
A complex number $z = x+iy$ can be represented as a point $P(x,y)$ in the XY-plane (Cartesian plane). This representation is called an Argand Diagram. We can plot complex numbers as points in the Cartesian coordinates with the `plot` function, and in polar coordinates with `compass` or `polar` functions.

Ejemplo 5 *Graphically represent the complex number $z = 3+2i$.*

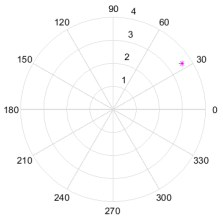
```
>> z=3+2i;
>> plot(z,'*m')
```




```
>> compass(z)
```




```
>> r=abs(z);
>> theta=angle(z);
>> polar(theta,r,'*')
```



Ejercicio 2 *Create a Matlab function named* `convert_polar.m` *that converts to polar a complex number z given in Cartesian form, and plot it.*

Ejercicio 3 *Create a Matlab function named* `zproduct.m` *that computes the product the quotient of two given complex numbers* z_1 *and* z_2.

Ejercicio 4 *Create a Matlab function named* `zpower.m` *that computes the* n^{th} *power of a complex number z..*

Ejercicio 5 *Create a Matlab function named* `zroots.m` *that computes the* n^{th} *roots of a complex number z, and plot them.*

2.3. Polynomial roots in $\mathbb{C}$

Ejemplo 6 *Compute the roots of the polynomial* $p(x) = x^3 + x + 1$*:*

```
>> p=[1 0 2 1];
>> roots(p)
```

We can see that for each complex root its conjugate is also a root, since the coefficients of the polynomial are real.

Ejercicio 6 *Create a function that:*

a) Computes the n^{th} *roots of z (for example, consider* $n = 8$ *y* $z = -1$*).*

b) Graphically represent them using plot, compass y polar.

c) Computes their sum and their product.