TEMA 3.- PROGRAMACIÓN EN **OCTAVE**

Asignatura: Informática Aplicada
Grado en Ingeniería de Materiales

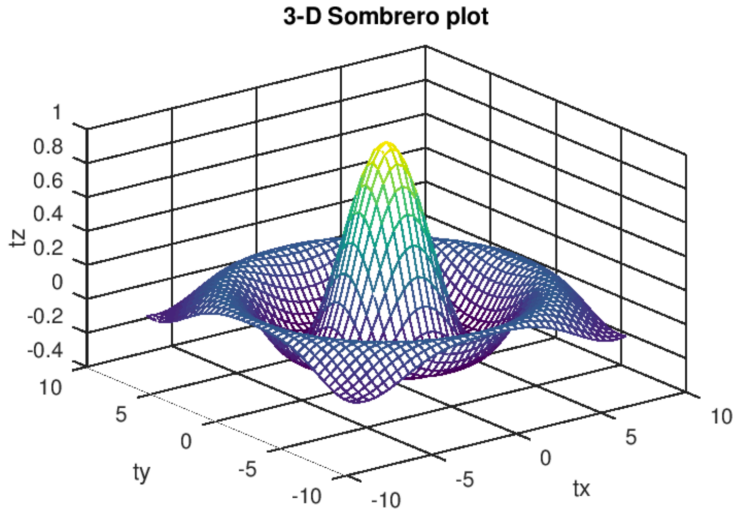
Hernán Santos Expósito
Departamento de Matemática Aplicada, Ciencia e Ingeniería de Materiales y Tecnología Electrónica
Departamental I Despacho 015 Campus de Móstoles.
hernan.santos@urjc.es

ÍNDICE

1.- INTRODUCCIÓN

¿Qué es Octave?

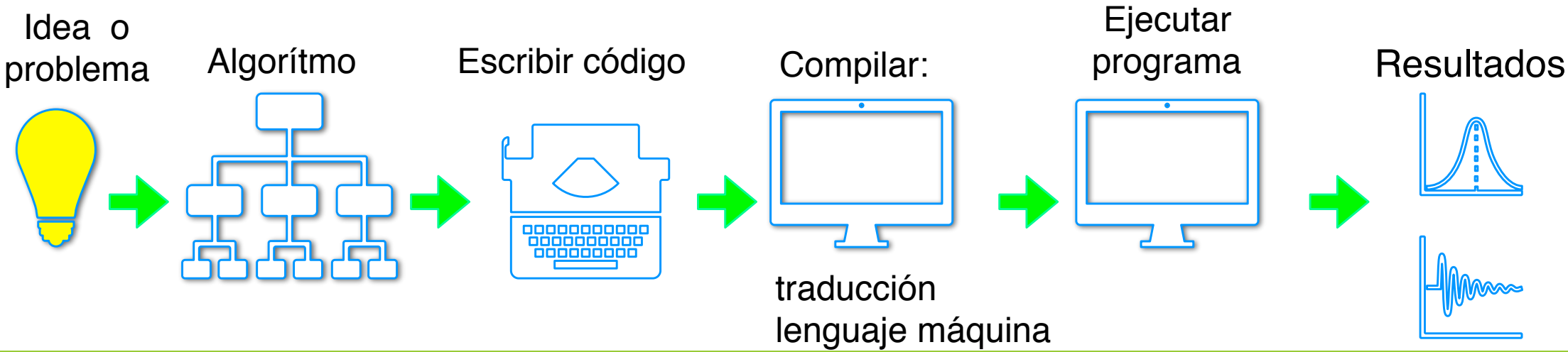
- Es un lenguaje de programación de **alto nivel** destinado principalmente a cálculos numéricos.
- Provee una **interface sencilla** para su uso.
- Usa un lenguaje compatible con **Matlab**, y también con módulos de otros lenguajes
- Es un software **libre** redistribuido.
- Es facilmente **extensible y personalizable**.
- Se puede usar par resolver (entre otros):
    - Problemas de álgebra lineal.
    - Integrar funciones ordinarias.
    - Integrar ecuaciones diferenciales ordinarias.



1.- INTRODUCCIÓN

Introducción a la programación. Lenguajes de programación:

- **Programación (informática)**: proceso por el cual una persona desarrolla un programa, valiéndose de una herramienta que le permite escribir el código en un lenguaje y de otra capaz de traducirlo a "lenguaje máquina" con el que el microprocesador lo resuelve. Fases:



1.- INTRODUCCIÓN

GNU Octave

Introducción a la programación. Lenguajes de programación:

- **Lenguajes de programación**: Es un lenguaje formal que le proporciona al programador la capacidad de programar una serie de instrucciones (o algorítmos) con el fin de controlar el comportamiento de un sistema informático.
- **Clasificación con respecto al nivel de abstracción:**

Lenguajes de bajo-nivel

- Más aproximados al código de máquina (0 1)
- Instrucciones ejercen control directo sobre el Hardware
- Más complicados de programar
- Uso: controladores de dispositivos, aplicaciones en tiempo real

Lenguajes de Alto-nivel

- Más aproximados al lenguaje humano o al de las matemáticas.
- Más sencillos y rápidos de usar
- Uso: ámbitos computacionales para crear todo tipo de aplicaciones y programas informáticos
- Ejemplos: C, C++, Fortran, Cobol, Java, Python, Octave

1.- INTRODUCCIÓN

GNU Octave

Introducción a la programación. Lenguajes de programación:

- **Ejemplos:**

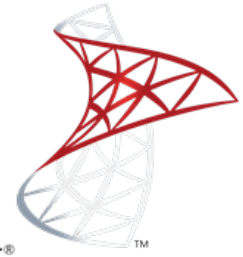
Ambito científico e ingeniería

Fortran

GNU Octave

Ambito Aplicaciones informáticas

Ambito ingeniería. Banca. Bases de datos



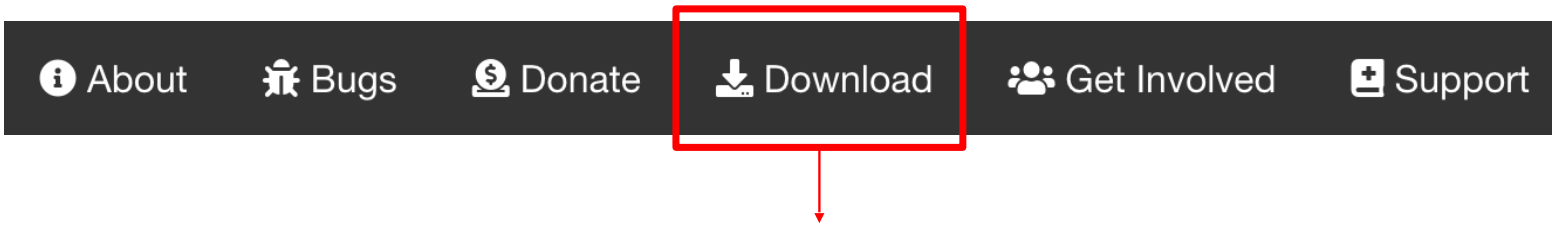
Microsoft SQL Server

1.- INTRODUCCIÓN

¿Cómo acceder a Octave?

- **Instalación en el propio ordenador personal:**

https://www.gnu.org/software/octave/index

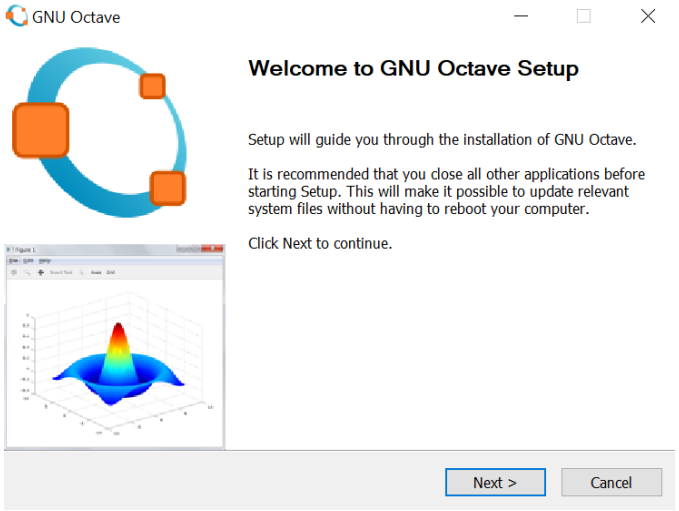


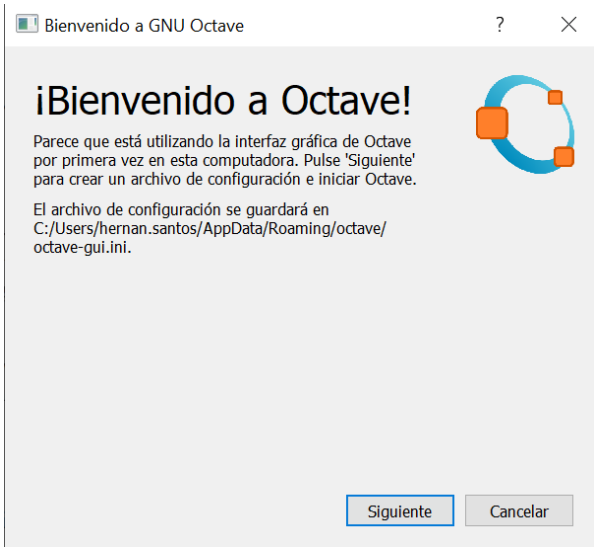
- Windows-64 (recommended)
  - octave-6.2.0-w64-installer.exe (~ 300 MB) [signature]
  - octave-6.2.0-w64.7z (~ 300 MB) [signature]
  - octave-6.2.0-w64.zip (~ 530 MB) [signature]

1.- INTRODUCCIÓN

¿Cómo acceder a Octave?

- **Instalación en el propio ordenador personal:**



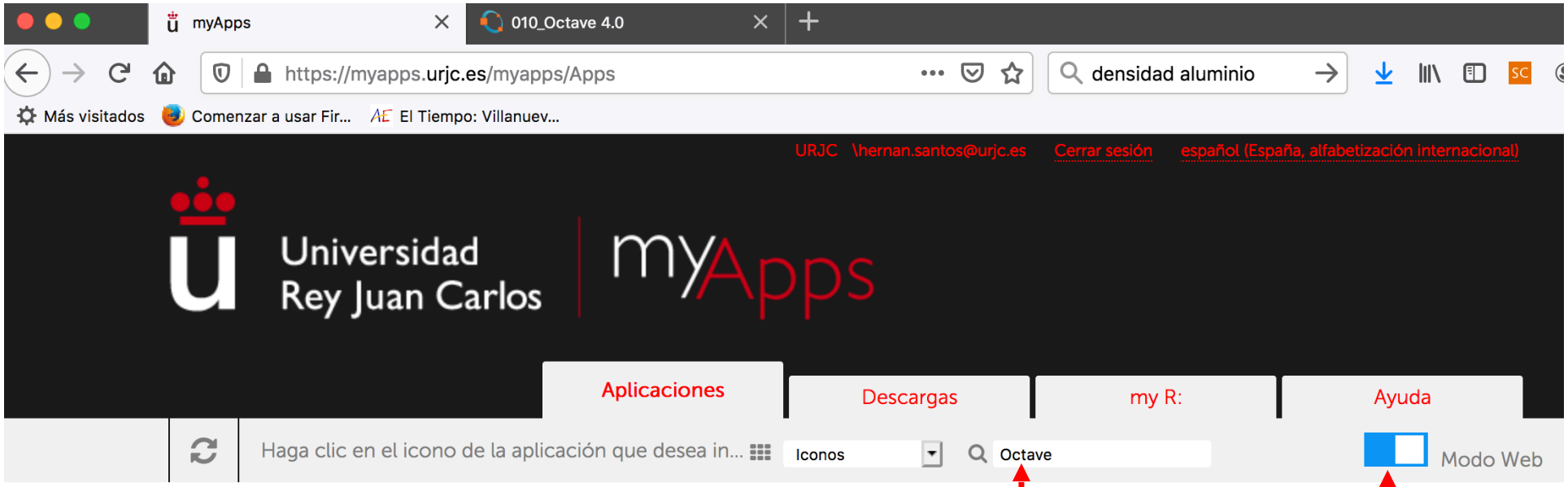


1.- INTRODUCCIÓN

¿Cómo acceder a Octave?

- **A través de Myapps (urjc)**

https://myapps.urjc.es/myapps



En Aplicaciones.- Clickar sobre el icono

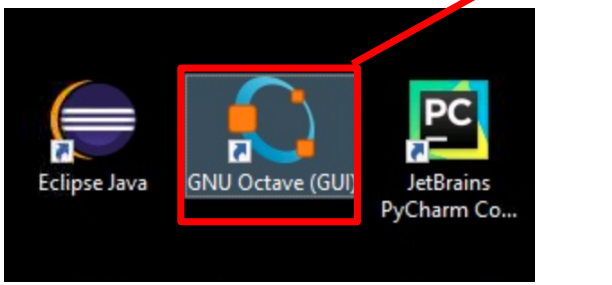
En Aplicaciones.- Buscar el programa Octave

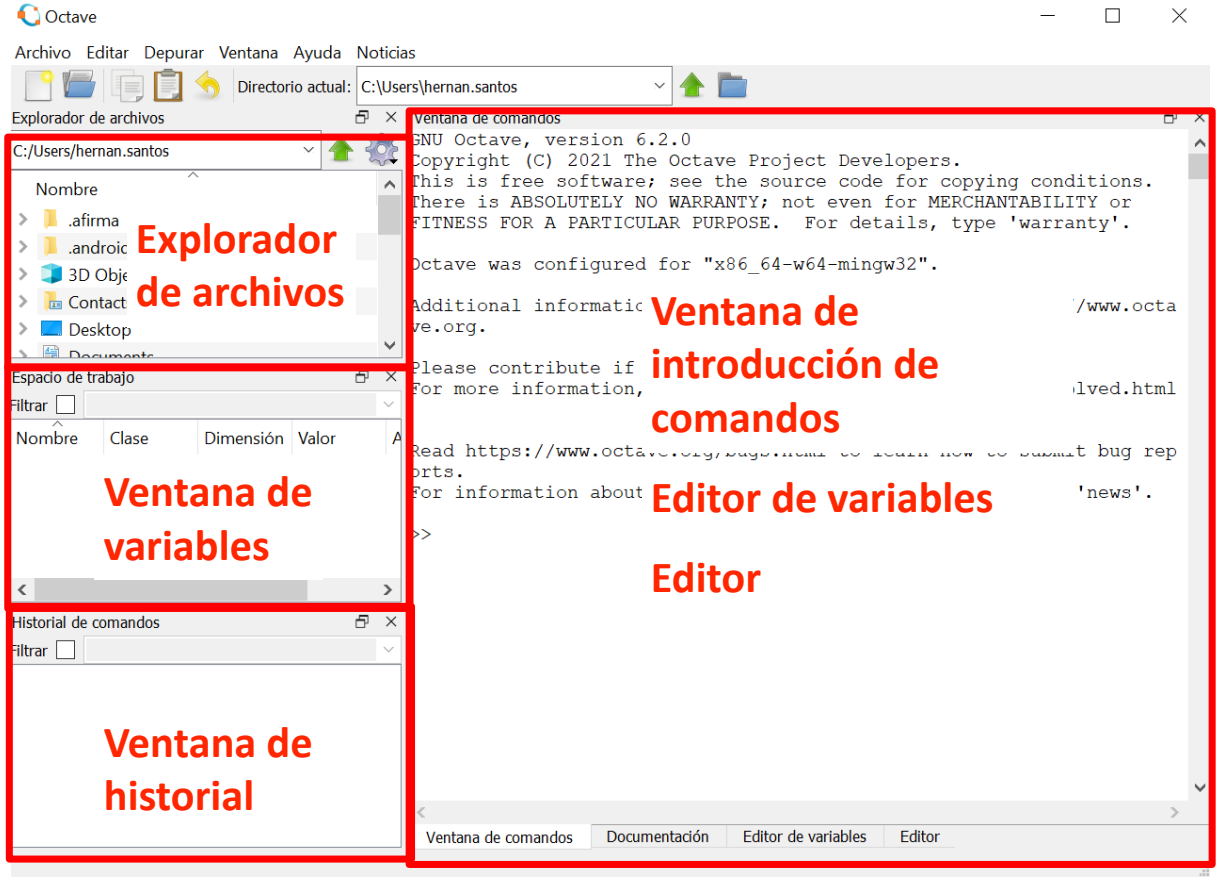
En Aplicaciones.- Seleccionar el Modo que deseáis En Modo Web (azul) se abre otra ventana de navegador.

1.- INTRODUCCIÓN

¿Cómo acceder a Octave?

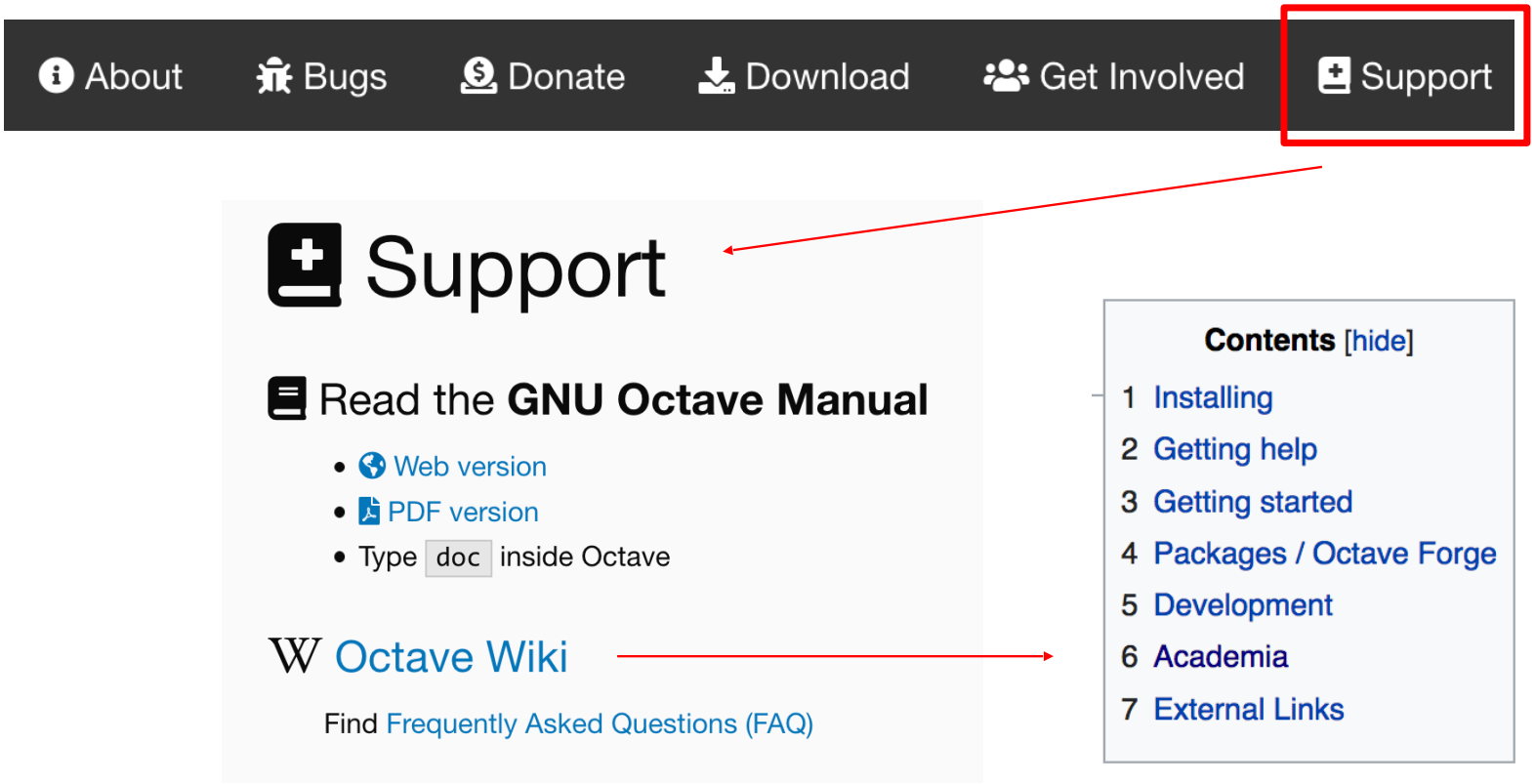
Paso 7.- Una vez seleccionado Octave aparece el escritorio remoto. Para acceder al programa Octave se selecciona el modo interactivo del programa GUI.





1.- INTRODUCCIÓN

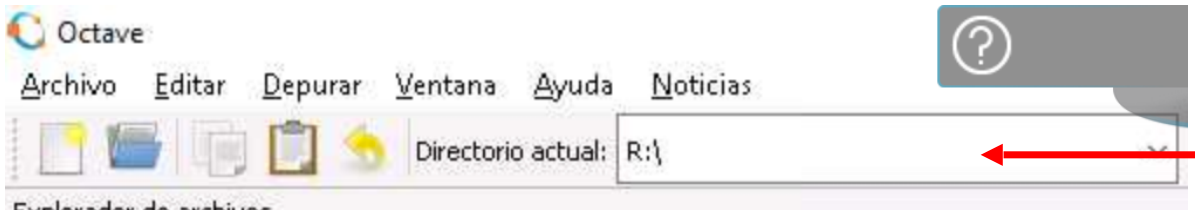
Manuales completos: https://www.gnu.org/software/octave/index



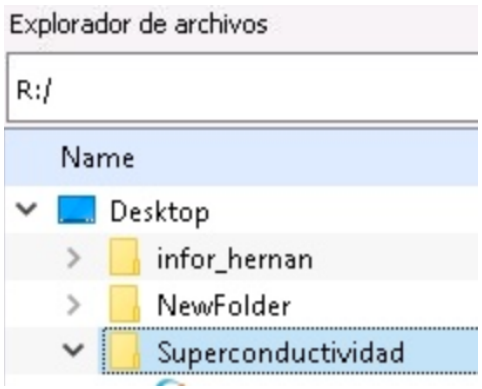
1.- INTRODUCCIÓN

Ejecutar programas

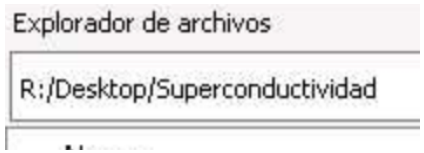
EJEMPLO R:/Desktop/Superconductivdad/
(que es donde se han colocado los ficheros del programa)



Teclear directamente en el *directorio actual*: R:



En explorador de archivos se selecciona el directorio que habéis creado donde se han introducido los archivos del programa. Esto se hace clickando dos veces sobre la carpeta aparecerá la ruta en el Explorador de archivos:



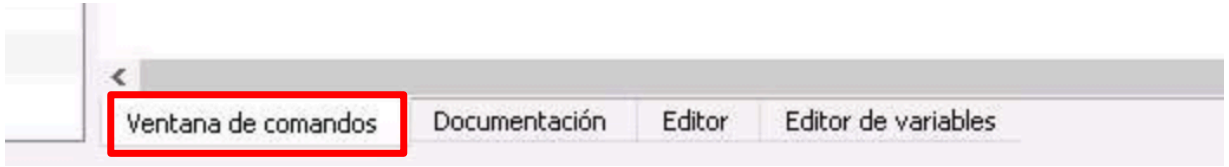
Y ya estaría listo para ejecutar los programas que tengan la terminación .m (octave) desde la ventana de comandos

1.- INTRODUCCIÓN

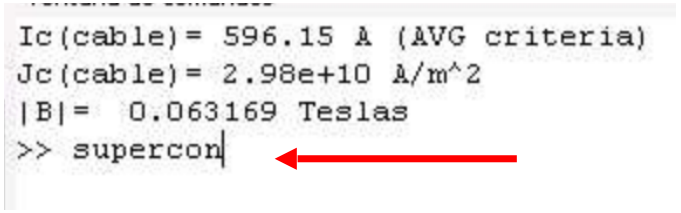
GNU Octave

Ejecutar programas

Una vez que estemos en la ruta Para ejecutar el programa nos situamos en la ventana de comandos y simplemente tecleamos el nombre del programa sin la extensión .m



Ejemplo si hemos construido el fichero supercon.m



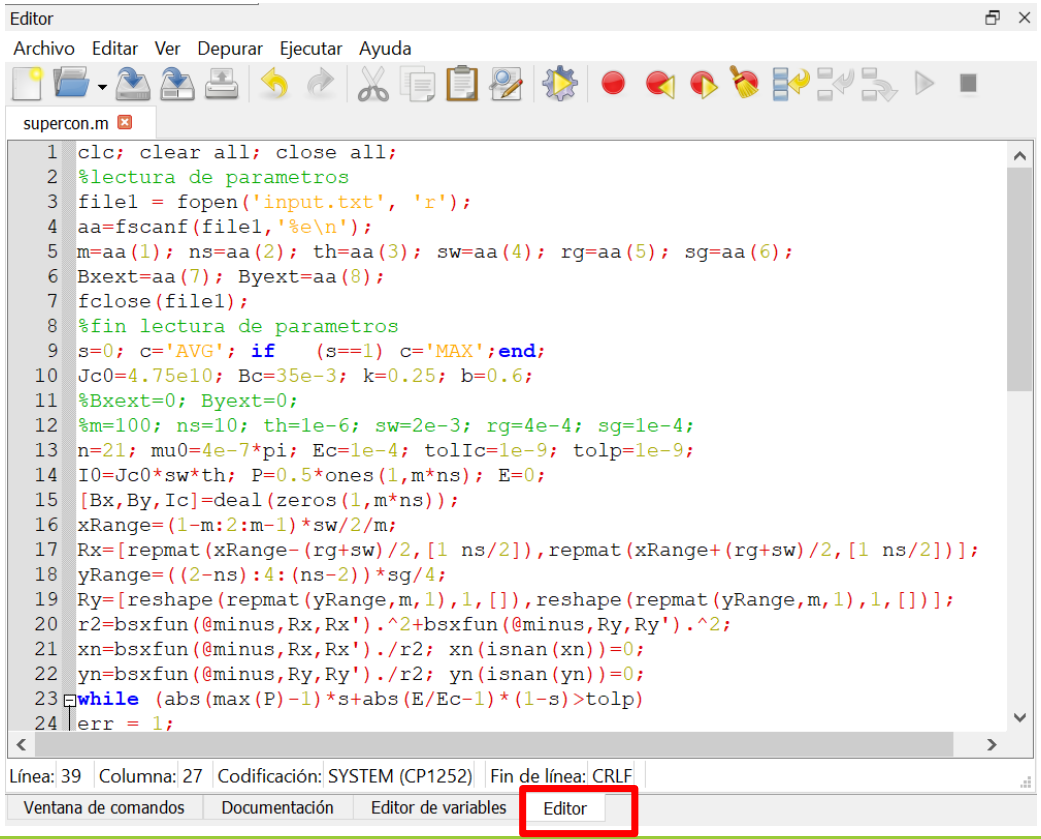
Y se ejecutará el programa supercon.m.

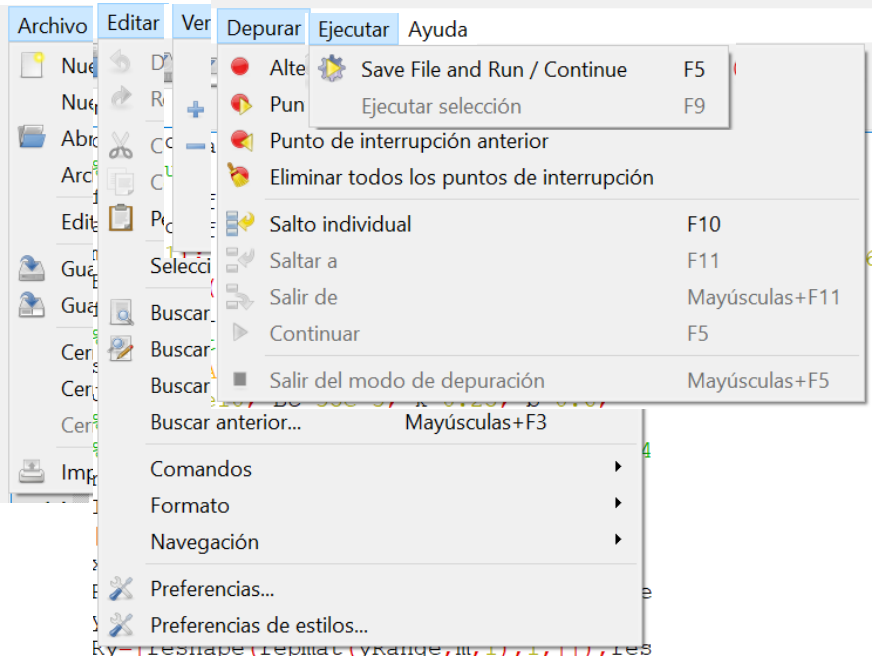
1.- INTRODUCCIÓN

GNU Octave

Abrir/modificar programas y ficheros

Para modificar, abrir y crear programas o ficheros utilizaremos la pantalla de editor





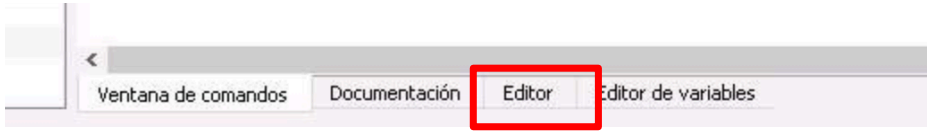
1.- INTRODUCCIÓN

GNU Octave

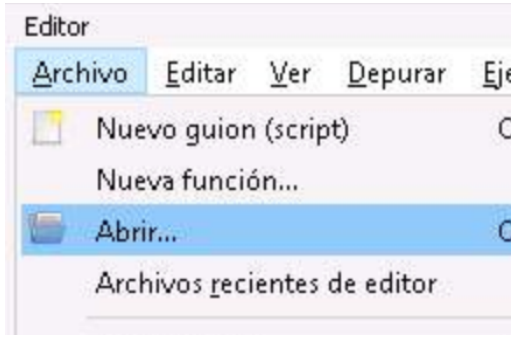
Abrir/modificar programas y ficheros

Ejemplo: abrir un archivo que no sea .m

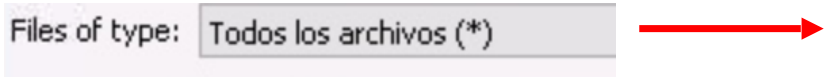
Primero abrimos la ventana del Editor por si tenemos que modificar el input del programa

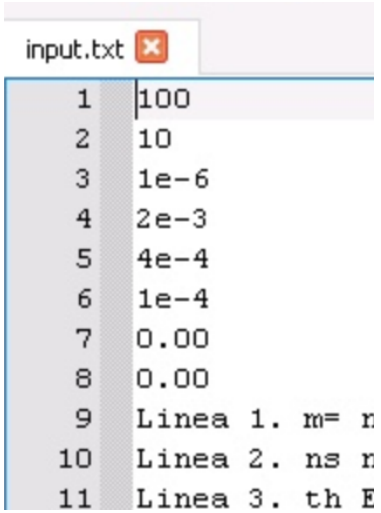


Dentro del Editor en el submenu desplegable Archivo abrimos el fichero input.txt



Seleccionamos que se abran todos los archivos y seleccionamos el fichero





Operaciones matemáticas

2.- Operaciones Matemáticas

GNU Octave

Operadores

Operación	Simbolo Octave
Suma	+
Resta	-
Multiplicación	*
División	/
Potencia	^
Raíz	sqrt()

Se puede usar Octave como calculadora escribiendo en la ventana de comandos las operaciones aritméticas y pulsaron sobre return.

Se obtiene el resultado a través de ans (answer.)

Prioridad en las Operaciones

Las operaciones no se realizan en el orden en el que están escritas, sino que siguen el siguiente orden:

1. Potencias.
2. Multiplicaciones y divisiones.
3. Sumas y Restas.
4. Dentro de cada grupo de izquierda a derecha.

Para modificar el orden se utilizan paréntesis ():

En caso de paréntesis, se calculan primero el ()

Si hay varios () anidados primero se calculan los más internos.

2.- Operaciones Matemáticas

GNU Octave

Operadores

```
>> 2*4
ans = 8
>> -2.1*3+2
ans = -4.3000
>> 8*6-2/4-1
ans = 46.500
>> sqrt(4)
ans = 2
>>
```

Prioridad en las Operaciones

```
>> 2+5*4
ans = 22
>> (2+5)*4
ans = 28
>> 2^4*1.5
ans = 24
>> 2^(4*1.5)
ans = 64
>> 2^2/2+3
ans = 5
>> 2^2/(2+3)
ans = 0.8000
>>
```

Ventana de comandos

2.- Operaciones Matemáticas

Operadores

Ejemplos:

$$\frac{5^3 + 4^2}{\frac{5}{\sqrt{2}} + \left(\frac{1}{2}\right)^{2/3}}$$

$$\frac{20^{\frac{3.7}{2}}}{\frac{2}{\sqrt{2}} + \left(\frac{1}{2^{1.2}}\right)^5}$$

Ventana de comandos

```
>> (5^2+4^2)/(5/sqrt(2)+(1/2)^(2/3))
ans = 9.8428
>> 20^(3.7/2)/((2/sqrt(2))+(1/2^(1.2))^5)
ans = 178.49
>>
```

2.- Operaciones Matemáticas

GNU Octave

Operadores lógicos verdadero: 1 falso: 0

<	→	Verdadero si es menor X<Y
>	→	Verdadero si es mayor X>Y
==	→	Verdadero si es igual X==Y
<=	→	Verdadero si es menor o igual X<=Y
>=	→	Verdadero si es mayor o igual X>=Y
!=	→	Verdadero si es diferente X!=Y
++	→	Suma 1, incrementa el valor de una variable en 1.

```
>> x=8
x = 8
>> y=10
y = 10
>> x<y
ans = 1
>> y<x
ans = 0
>> x==y
ans = 0
```

```
>> x<=y
ans = 1
>> x>=y
ans = 0
>> x!=y
ans = 1
>> x++
ans = 8
>> x
x = 9
```

2.- Operaciones Matemáticas

GNU Octave

Operadores lógicos

verdadero: 1 falso: 0

||

Operador lógico "or": devuelve verdadero si alguna de varias opciones es verdadera. Se utiliza en bucles y comparaciones para comprobar varias condiciones simultáneas.

x < 10 || y > 3

En este ejemplo se obtiene verdadero si se cumple que x < 10 ó y > 3.

&&

Operador lógico "and": devuelve verdadero si se cumplen varias opciones simultáneamente. Se utiliza en bucles y comparaciones para comprobar varias condiciones simultáneas.

x < 10 && y > 3

En este ejemplo se obtiene verdadero si se cumple que x < 10 y y > 3 simultáneamente.

Ventana de comandos

```
>> x=8
x = 8
>> y=2
y = 2
>> x<10 || y>3
ans = 1
>> x<10 && y>3
ans = 0
>> x<10 && y>=3
ans = 0
>> x<10 && y>=2
ans = 1
>>
```

2.- Operaciones Matemáticas

GNU Octave

Definición y Asignación de variables

- Def. variable: “Sitio” en el memoria del sistema del sistema para guardar datos.
- El signo IGUAL (=) es una asignación

x = 8 ⟶ Asignamos el valor 8 a la variable x

y = 3 + 2 ⟶ Asignamos el valor de la operación 3 + 2 a la variable y.

z = x + y ⟶ Asignamos el valor de la operación x + y a la variable z.

```
>> x=8
x = 8
>> y=3+2
y = 5
>> z=x+y
z = 13
>> x=8;
>> y=5
y = 5
>> z=2*x+y/3
z = 17.667
>>
```

Características de las variables:

- **Siempre hay que definir las variables antes de usarlas.**
- Las variables son sensibles a mayúsculas y minúsculas. No es lo mismo la variable X que x.
- Los nombres de variable deben ser CORTOS.
- Los nombres de variable no deben empezar por un un número

2.- Operaciones Matemáticas

GNU Octave

Definición y Asignación de variables

- Algunas variables predefinidas:

Variables	Función
ans	Variable del sistema para almacenar el resultado de evaluar expresiones
pi	Número π
inf	Infinito; número más grande que se puede almacenar
NaN	Magnitud no numérica; resultado de cálculos indefinidos.
eps	Precisión relativa en punto flotante (2.2204e-016)
realmin	Número real positivo más pequeño utilizable (2.2251e-308)
realmax	Número real positivo más grande utilizable (1.7977e308)
i, j	Unidad imaginaria; raíz de -1.

2.- Operaciones Matemáticas

GNU Octave

Comandos

format : define el formato de las variables

Format	Significado
Short	Parte entera + 6 decimales
Longe	Parte entera + 14 decimales
Bank	Parte entera + 2 decimales
Rat	Formato de fracciones (a/b)
Hex	Formato hexadecimal (0-9;A-F)
‘ ’	Definen una cadena de texto

```
>> format short
>> x=2/33
x = 0.060606
>> format longe
>> x
x = 6.060606060606061e-02
>> format bank
>> x
x = 0.06
>> format rat
>> x
x = 2/33
>> format hex
>> x
x = 3faf07c1f07c1f08
>> x='abcd'
x = abcd
>>
```

2.- Operaciones Matemáticas

GNU Octave

Comandos

Otros comandos útiles

Format	Significado
clc	Limpia la ventana de comandos
clear	Borra el valor de todas las variables
Clear x	Borra el valor de la variable x
disp()	Muestra lo que está entre paréntesis
who	Muestra la lista de variables
;	Indica find de fila y se usa para crear columnas Impide que se muestre el valor de una variable.

disp('hola') → Muestra la palabra hola

disp(hola) → Muestra el valor de la variable hola

```
>> format bank
>> hola=6
hola = 6.00
>> disp('hola')
hola
>> disp(hola)
6.00
>>
```

2.- Operaciones Matemáticas

GNU Octave

Funciones matemáticas elementales

Variables	Significado
exp(x)	Exponencial: e^x
log(x)	Logaritmo neperiano: ln(x)
log10(x)	Logaritmo decimal: $\log_{10}(x)$
abs(x)	Valor absoluto: |x|
sign(x)	Devuelve el signo del argumento x

Ventana de comandos

```
>> x=2^2
x = 4.00
>> a=exp(x)
a = 54.60
>> b=log(a)
b = 4.00
>> x=100
x = 100.00
>> c=log10(x)
c = 2.00
>> x=-3
x = -3.00
>> abs(x)
ans = 3.00
>> d=sign(x)
d = -1.00
>>
```

2.- Operaciones Matemáticas

GNU Octave

Funciones matemáticas elementales

Variables	Significado	Variables	Significado
sin(x)	Seno	asin(x)	Arco-seno
cos(x)	Coseno	acos(x)	Arco-coseno
tan(x)	Tangente	atan(x)	Arco-tangente
sec(x)	Secante	asec(x)	Arco-secante
csc(x)	Cosecante	acsc(x)	Arco-cosecante
cot(x)	Cotangente	acot(x)	Arco-cotangente

Ventana de comandos

```
>> a=sin(pi)
a = 0.00
>> a=sin(pi/2)
a = 1.00
>> a=cos(pi)
a = -1.00
>> a=tan(pi/6)
a = 0.58
>> sec(pi)
ans = -1.00
>> csc(pi)
ans = 8165889364191922.00
>> cot(pi)
ans = -8165619676597685.00
>> asin(1)
ans = 1.57
>> acos(1)
ans =   0
>> acos(-1)
ans = 3.14
>> atan(0.5)
ans = 0.46
>>
```

2.- Operaciones Matemáticas

GNU Octave

Números Complejos

- Se usa la forma binómica para representar un número complejo:

$$z = a + bi$$

Donde *a* es la parte real, *b* la imaginaria e $i = j = \sqrt{-1}$

Variables	Significado
z=complex(a,b)	Definir número complejo
abs(z)	Módulo de un número complejo
angle(z)	Argumento de un número complejo
conj(z)	Conjugado de un número complejo
imag(z)	Parte imaginaria de un número complejo
real(z)	Parte real de un número complejo

- Froma exponencial

$$z = \rho \exp^{i\theta}$$

$$\rho = \sqrt{a^2 + b^2}$$

$$\theta = \tan^{-1}(a/b)$$

2.- Operaciones Matemáticas

GNU Octave

Números Complejos

- Ejemplos:

Formas de expresar los números complejos

Ventana de comandos

```
>> z=4+6j
z =  4 + 6i
>> rho=abs(z)
rho = 7.211102550927978
>> theta=angle(z)
theta = 0.982793723247329
>> Ze=rho*exp(j*theta)
Ze =  4 + 6i
>> Zp=rho*(cos(theta)+j*sin(theta))
Zp =  4 + 6i
>>
```

Operaciones con n. complejos:

```
>> z1=1+5j
z1 =  1 + 5i
>> z2=2+2j
z2 =  2 + 2i
>> z1-z2
ans = -1 + 3i
>> z3=z1*z2
z3 =  -8 + 12i
>> z4=z1/z2
z4 =  1.500000000000000 + 1.000000000000000i
>> z5=z1^3
z5 =  -74 - 110i
>>
```

2.- Operaciones Matemáticas

Números Complejos

- Ejercicio: Se considera $z_1 = 4 + 5i,\ \ z_2 = 4 + i$. Determinar, con Octave, los siguientes apartados:
  - Realizar las siguientes operaciones:

$$z_1 + z_2,\ z_1 \cdot z_2,\ \bar{z_1},\ \frac{z_1}{z_2},\ z_1^2$$

  - Calcular el módulo y el argumento de z_1
  - Escribir la forma trigonométrica y exponencial de z_1
  - Calcular $\sin(z_1)$ y $\cos(z_1)$

```
>> z1=4+5j
z1 =  4 + 5i
>> z2=4+i
z2 =  4 + 1i
>> z1+z2
ans =  8 + 6i
>> z1*z2
ans =  11 + 24i
>> conj(z1)
ans =  4 - 5i
>> z1/z2
ans =  1.235294117647059 + 0.941176470588235i
>> z1^2
ans =  -9 + 40i
```

```
>> rho=abs(z1)
rho = 6.403124237432849
>> theta=angle(z1)
theta = 0.896055384571344
>> z1_tri=rho*(cos(theta)+i*sin(theta))
z1_tri =  4.000000000000000 + 4.999999999999999i
>> z1_polar=rho*exp(i*theta)
z1_polar =  4.000000000000000 + 4.999999999999999i
>> sin(z1)
ans = -56.16227422023235 - 48.50245524177091i
>> cos(z1)
ans = -48.50685945784458 + 56.15717492513018i
>>
```

Vectores y Matrices

3.- Vectores y Matrices

GNU Octave

Vectores

- Un vector-fila de dimensión n se define escribiendo sus componentes entre corchetes [] y separándolos por comas o espacios en blanco:

```
>> v1=[1,2,-3,1.1]
v1 =

   1.0000   2.0000  -3.0000   1.1000
```

  - Un vector-columna se crea de la misma forma pero separando los componentes por punto y coma, ;

```
>> v1=[1;2;-3;1.1]
v1 =

   1.0000
   2.0000
  -3.0000
   1.1000

>>
```

  - También podíamos haber creado el vector-columna a partir del vector fila simplemente realizando su transpuesta. Con la ‘ podemos realizarlo de la siguiente manera:

```
>> v1=[1,2,3,4]
v1 =

   1   2   3   4

>> v1_tras=v1'
v1_tras =

   1
   2
   3
   4
```

- Se puede acceder a los elementos de un vector seleccionando entre paréntesis la posición del elemento

```
>> v1=[1,-1,2,2.5]
v1 =

   1.0000  -1.0000   2.0000   2.5000

>> v1(4)
ans = 2.5000
>> v1(2)
ans = -1
```

3.- Vectores y Matrices

GNU Octave

Vectores

- Se pueden crear vectores automáticamente utilizando los dos puntos:

A=2:8 → A=[2,3,4,5,6,7,8] → Se crea un vector automáticamente con inicio 2 y final en 8

A=2:3:8 → A=[2,5,8] → Se crea un vector que empieza en 2 y termina en 8 espaciado 3 elementos

A=2:3:7 → A=[2,5] → Como el espaciado de elementos no lo permite queda un vector de 2 elementos

```
>> A=2:8
A =

   2   3   4   5   6   7   8
```

```
>> A=2:3:8
A =

   2   5   8
```

```
>> A=2:3:7
A =

   2   5
```

3.- Vectores y Matrices

GNU Octave

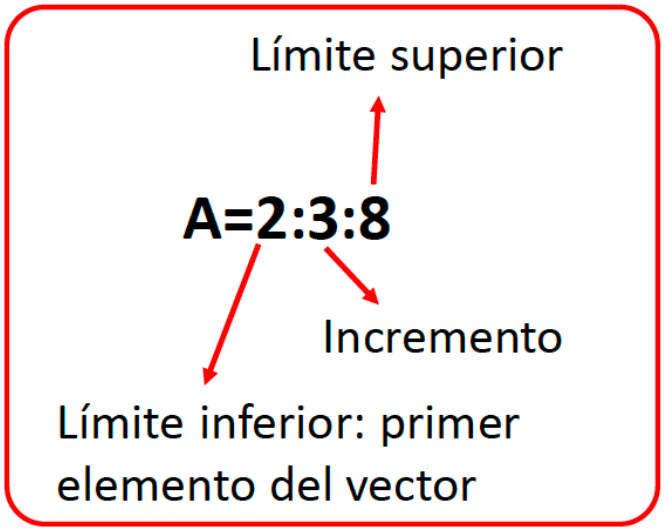
Vectores

- Se pueden crear vectores automáticamente utilizando los dos puntos:

A=2:8 → A=[2,3,4,5,6,7,8]

A=2:3:8 → A=[2,5,8]

A=2:3:7 → A=[2,5]



- **linspace()** Otra función de generación de vectores: Genera vectores con elementos espaciados igualmente

linspace(2,10,5) → [2,4,6,8,10]

Genera un vector de 5 elementos que comienza en 2 y termina en 10

3.- Vectores y Matrices

GNU Octave

Vectores

- **Funciones específicas para vectores:**

Funciones	Significado
length(v)	Devuelve el número de componentes del vector v
max(v), min(v)	Devuelve el valor máximo/mínimo entre las componentes de v
sum(v), prod(v)	Devuelve la suma/producto de las componentes de v
norm(v)	Devuelve el módulo del vector v
dot(v1,v2)	Devuelve el vector producto escalar de v1 y v2
cross(v1,v2)	Devuelve el vector producto vectorial de v1 y v2
sort(v)	Devuelve un vector con las componentes de v ordenadas de menor a mayor

3.- Vectores y Matrices

GNU Octave

Vectores

- **Funciones específicas para vectores:**

Definimos los vectores:

```
>> v1=1:3
v1 =

   1   2   3

>> v2=[1,0,2]
v2 =

   1   0   2
```

Aplicamos las funciones, longitud, máximo, mínimo, suma y producto.

```
>> a=length(v1)
a = 3
>> b=max(v1)
b = 3
>> c=min(v1)
c = 1
>> d=sum(v1)
d = 6
>> e=prod(v1)
e = 6
```

Aplicamos norma, producto escalar y vectorial

```
>> f=norm(v1)
f = 3.7417
>> g=dot(v1,v2)
g = 7
>> v3=cross(v1,v2)
v3 =

   4   1  -2

>> v4=sort(v1)
v4 =

   1   2   3
```

3.- Vectores y Matrices

GNU Octave

Matrices

- Las matrices se definen introduciendo las filas como vectores-fila y introduciendo nuevas filas mediante punto y como ; o saltos de línea

```
>> A=[1,3,5;2,4,6;9,9.1,9.2]
A =

   1.0000   3.0000   5.0000
   2.0000   4.0000   6.0000
   9.0000   9.1000   9.2000
```

- Otra forma de crear matrices es concatenando diferentes vectores o combinando matrices con vectores utilizando los corchetes []

```
>> B=[1;2;3]
B =

   1
   2
   3

>> C=[A,B]
C =

   1.0000   3.0000   5.0000   1.0000
   2.0000   4.0000   6.0000   2.0000
   9.0000   9.1000   9.2000   3.0000
```

3.- Vectores y Matrices

GNU Octave

Matrices

- También como los vectores se pueden crear de forma automática empleando los dos puntos:

```
>> A=[1:3;1:3:9;2,3,4]
A =

   1   2   3
   1   4   7
   2   3   4
```

- O realizar una operación de transponer la matriz con ’

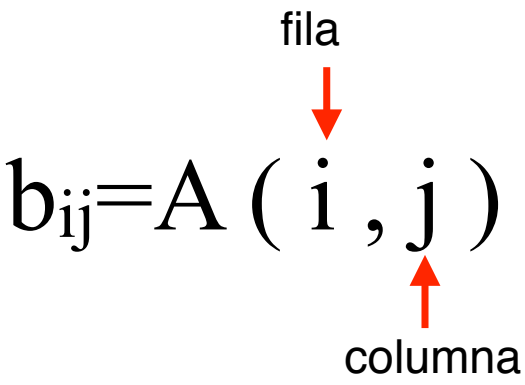
```
>> B=A'
B =

   1   1   2
   2   4   3
   3   7   4
```

3.- Vectores y Matrices

GNU Octave

Matrices

- Para extraer el valor del cualquier elemento se pone el nombre de la variable usada para la matriz y entre paréntesis, la fila y la columna separada por una coma.




```
>> A=[1,2,3;4,5,6;7,8,9]
A =

   1   2   3
   4   5   6
   7   8   9

>> b=A(1,1)
b = 1
>> b=A(2,1)
b = 4
>> b=A(3,2)
b = 8
>>
```

- Se puede extraer parte de una matriz utilizando los dos puntos:

- Una columna:

```
>> b=A(:,3)
b =

   3
   6
   9
```

- Una fila:

```
>> c=A(3,:)
c =

   7   8   9
```

- Una parte de una matriz:

```
>> d=A(2:3,2:3)
d =

   5   6
   8   9
```

3.- Vectores y Matrices

GNU Octave

Matrices

- Funciones predefinidas para construir matrices

Función	Significado
eye(n)	Forma la matriz identidad cuadrada de dimensión n
zeros(n)	Forma una matriz de ceros cuadrada de dimensión n
zeros(m,n)	Forma una matriz de ceros de m filas y n columnas
ones(n)	Forma una matriz de unos cuadrada de dimensión n
ones(m,n)	Forma una matriz de unos de m filas y n columnas

```
>> eye(3)
ans =

Diagonal Matrix

   1   0   0
   0   1   0
   0   0   1
```

```
>> zeros(2)
ans =

   0   0
   0   0
```

```
>> zeros(2,4)
ans =

   0   0   0   0
   0   0   0   0
```

```
>> ones(2)
ans =

   1   1
   1   1
```

```
>> ones(2,4)
ans =

   1   1   1   1
   1   1   1   1
```

3.- Vectores y Matrices

GNU Octave

Matrices

- Otras funciones importantes

Función	Significado
size(A)	Da el tamaño de la matriz [nºfilas nºcolumnas].
inv(A)	Calcula la inversa de la matriz A.
det(A)	Calcula el determinante de la matriz A.
trace(A)	Calcula la traza de la matriz A.
‘	Calcula la adjunta de la matriz A.

size()

```
>> A=[2:4;3:5;4:6;5:7]
A =

   2   3   4
   3   4   5
   4   5   6
   5   6   7
```

```
>> b=size(A)
b =

   4   3
```

- Admite otro parámetro: cuando vale 1 nos devuelve el nº de filas, y si vale 2 el nº de columnas

```
>> size(A,1)
ans = 4
>> size(A,2)
ans = 3
>>
```

3.- Vectores y Matrices

GNU Octave

Matrices

- Otras funciones importantes

inv()

$$inv(A) = \frac{1}{det(A)} A^T$$

```
>> A=[1,0,1;0,1,0;2,1,1]
A =

   1   0   1
   0   1   0
   2   1   1
```

```
>> B=inv(A)
B =

  -1  -1   1
   0   1   0
   2   1  -1
```

det()

```
>> D=det(A)
D = -1
```

trace()

```
>> E=trace(A)
E = 3
```

Transpuesta ‘

```
>> C=A'
C =

   1   0   2
   0   1   1
   1   0   1
```

3.- Vectores y Matrices

Matrices

- Operaciones con matrices

Operación	Simbolo Octave
Suma	+
Resta	-
Producto	*
Producto elemento a elemento	.*
Exponenciación	^
Exponenciación elemento a elemento	.^
División	/
División elemento a elemento	./
División invertida	\
División invertida elemento a elemento	.\

3.- Vectores y Matrices

GNU Octave

Matrices

- Operaciones con matrices. Ejemplos

Matrices

```
>> A=[1,1;0,-1]
A =

   1   1
   0  -1

>> B=[2,-1;0,1]
B =

   2  -1
   0   1
```

```
>> C=A+B
C =

   3   0
   0   0

>> D=A-B
D =

  -1   2
   0  -2

>> E=A*B
E =

   2   0
   0  -1
```

```
>> F=A.*B
F =

   2  -1
   0  -1

>> G=A^2
G =

   1   0
   0   1

>> G=B^2
G =

   4  -3
   0   1

>> H=A.^2
H =

   1   1
   0   1
```

```
>> H=B.^2
H =

   4   1
   0   1

>> I=A/B
I =

   0.5000   1.5000
        0  -1.0000

>> I=B/A
I =

   2   3
   0  -1

>> I=B./A
I =

     2    -1
   NaN    -1
```

```
>> J=A\B
J =

   2   0
   0  -1

>> J=A.\B
J =

     2    -1
   NaN    -1
```

3.- Vectores y Matrices

GNU Octave

Matrices

- **Aplicación de funciones a matrices**

- En general, si aplicamos una función a una matriz o a un vector nos devolverá una matriz o un vector con las mismas dimensiones con la función aplicada elemento a elemento.

```
>> A=[1,0,1;0,1,0;0.5,2/3,0.1]
A =

   1.0000         0   1.0000
        0   1.0000         0
   0.5000   0.6667   0.1000

>> B=cos(A)
B =

   0.5403   1.0000   0.5403
   1.0000   0.5403   1.0000
   0.8776   0.7859   0.9950
```

3.- Vectores y Matrices

GNU Octave

Matrices

- **Aplicación de operadores lógicos a matrices**

- Se pueden aplicar los operadores lógicos (&, xor) o relacionarles (<, >, <=, >=, ==, ~=) a las matrices o vectores y el resultado será una matriz o vector de verdadero o falso, con ceros o unos

```
>> A=[1,-1;2,0]
A =

   1  -1
   2   0

>> B=[1,2;-1,2]
B =

   1   2
  -1   2
```

```
>> C=A&B
C =

  1  1
  1  0
```

```
>> D=A>B
D =

  0  0
  1  0
```

```
>> D=A>=2
D =

  0  0
  1  0
```

- Se pueden mostrar los elementos que han dado valores positivos (unos) de la matriz original de la siguiente manera:

```
>> E=A(D)
E = 2
```

```
>> E=A(C)
E =

   1
   2
  -1
```

Mostrar y pedir información al usuario

4.- Interactuar con el usuario

disp() → Visto anteriormente

fprintf() →

- Permite un mayor control de las salidas por pantalla de lo que lo hace disp.
- Sirve para combinar texto y contenido de variables.
- Permite especificar el formato con el que se mostrarán los valores.
- La sintaxis es la siguiente:

fprintf (formato, expresiones)

Formato: contiene el texto y las especificaciones de formato para las salidas
Expresiones: variables que se quiere visualizar separadas por comas.

4.- Interactuar con el usuario

fprintf() → **Ejemplo**

```
x=5
fprintf(´x = %d m \n´,x)
```

Salto de línea

Va a mostrar: **x=5 m**

(sin decimales)

Opciones de salida para las variable

Variable a mostrar

Opciones:

- **%d** → Indica que la variable es entera (sin decimales)
- **%f** → Indica que la variable se mostrará con decimales
- **%e** → Indica que la variable se mostrará en notación exponencial
- **%s** → Indica que la variable se mostrará en texto

Va a marcar la posición donde se mostrará la variable dentro del texto

4.- Interactuar con el usuario

GNU Octave

fprintf()

%f Podemos especificar el número de posiciones y los decimales

%7.2f → Especificará que se reserven 7 posiciones con dos decimales

%.2f → Especificará dos decimales

```
>> x=5
x = 5
>> y=2
y = 2
>> z=1
z = 1
```

```
>> fprintf('La base es b=%.2f metros,
la altura es a=%d m, y el fondo es f=%e m\n',x,y,z)

La base es b=5.00 metros, la altura es a=2 m,
a es a=2 m, y el fondo es f=1.000000e+00 m
```

4.- Interactuar con el usuario

GNU Octave

input()

Sirve para pedir al usuario que introduzca información

Ejemplo:

a=input(‘Dame un número ‘) { Va a mostrar el texto: 'Dame un número’ y va a esperar a que introduzcamos uno El valor que pongamos se asignará a la variable a.

Si queremos que lo que se introduzca sea un texto:

a=input(‘Dame un nombre’,’s’)

Indica que lo que el usuario va a introducir es una cadena de teto

Si el comando “input” no tiene el segundo argumento ’s' hay que introducir un número

4.- Interactuar con el usuario

GNU Octave

Ejemplo completo:

```
a=input('Teclee el lado del cuadrado ');
b=input('¿En qué unidades está? ','s');
c=x^2;
fprintf('El área es %.2f %s\n',c,b)
```

Se pone el ; para que no salga el valor por pantalla.

```
>> a=input('Teclee el lado del cuadrado ');
Teclee el lado del cuadrado 4

>> b=input('En que unidades esta ','s');
En que unidades esta metros

>> c=a^2;
>> fprintf('El area es %.2f %s^2 \n',c,b)
El area es 16.00 metros^2
```

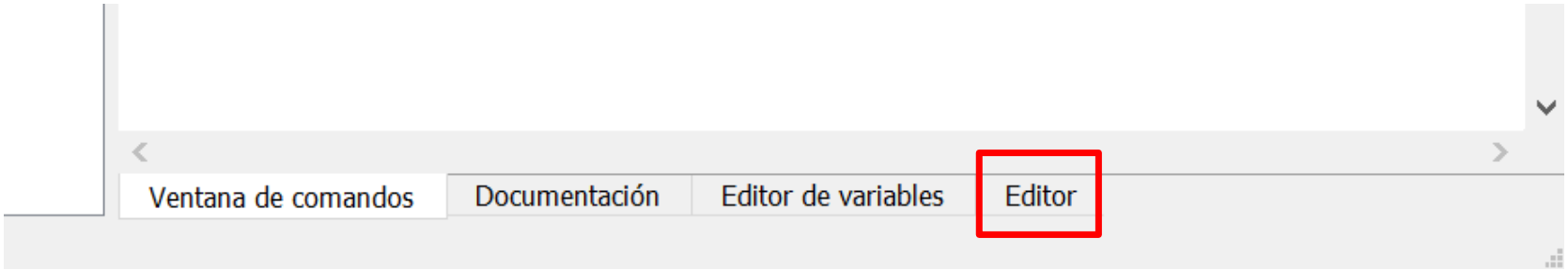
Diseñar un Programa

5.- Diseñar un programa

En el diseño de un programa se pueden seguir los siguientes pasos:

- Definir bien el problema.
- Identificar las entradas de datos.
- Identificar las salidas de datos.
- Definir el algoritmo a implementar.

Para crear el programa necesitamos un EDITOR de texto, el cual viene incorporado en una de las pestañas de Octave.



5.- Diseñar un programa

Para escribir el programa se pueden seguir los siguientes pasos:

1. Definir las variables que se necesitan.
2. Pedir al usuario que introduzca valores.
3. Operar con las variables.
4. Dar los resultados.

Otras directrices:

- Se suelen incorporar **comentarios** que hacen más fácil modificar el programa si lo tenemos que retomar más adelante. Esto se hace mediante los símbolos **% ó #.**
- Al final de cada linea se suele poner **;** para que no saque en pantalla todas las definiciones de variables o cada cálculo.
- Es conveniente hacer el uso de funciones que veremos más adelante en programas largos.

5.- Diseñar un programa

GNU Octave

Ejemplos 1: Diseñar, escribir y ejecutar un programa donde se pida el radio de un circulo, las unidades y calcule el área que deberá de sacar por pantalla.

Una vez que tegamos claro lo que hay que hacer, escribimos el programa:

area_circulo.m

```
1  % Programa calculo area de un circulo
2  %
3  % Definición de variables. formato.
4  %
5  % Entrada de datos
6  %
7  a=input('Radio del circulo=');
8  unit=input('en que unidades? ','s');
9  %
10 % algoritmo de calculo
11 %
12 Area=pi*a^2;
13 %
14 % muestra el resultado.
15 %
16 fprintf('El area es=%.2f %s^2 \n',Area,unit);
```

Información del programa

No hay variables que definir. El formato que seguirán será el de doble precisión

Pedimos datos al usuario. Radio del círculo y en qué unidades está.

Introducimos el algoritmo de cálculo.

Sacamos por pantalla el resultado.

Más adelante veremos como guardar los resultados en un fichero

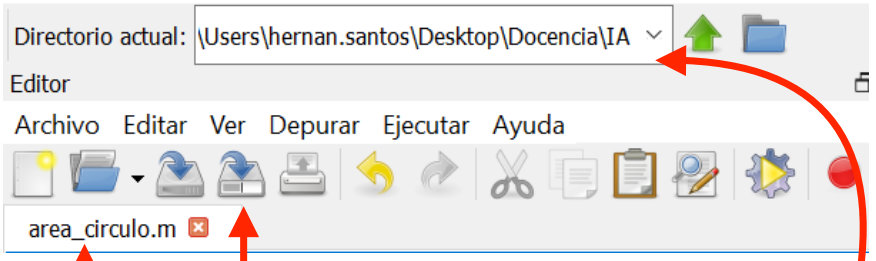
5.- Diseñar un programa

Ejemplo 1: Diseñar, escribir y ejecutar un programa donde se pida el radio de un circulo, las unidades y calcule el area que deberá de sacar por pantalla.

area_circulo.m

```
% Programa calculo area de un circulo
%
% Definición de variables. formato.
%
% Entrada de datos
%
a=input('Radio del circulo=');
unit=input('en que unidades? ','s');
%
% algoritmo de calculo
%
Area=pi*a^2;
%
% muestra el resultado.
%
fprintf('El area es=%.2f %s^2 \n',Area,unit);
```

Guardamos el programa



Pulsamos en este icono que es “guardar como”. Aseguraros que se guarda en el directorio donde estéis trabajando.

Aparecerá el nombre con el lo guardáis

area_circulo.m

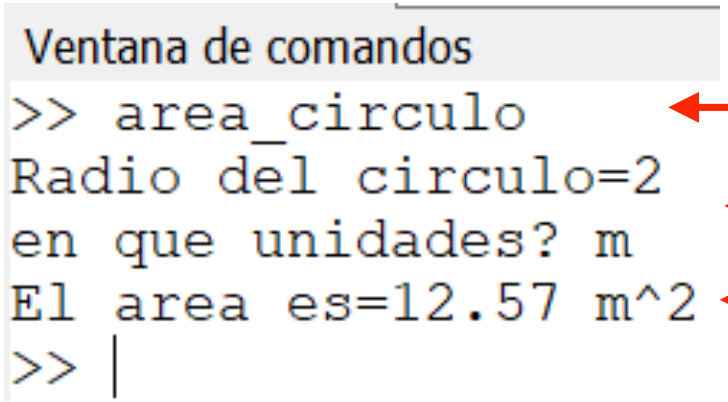
5.- Diseñar un programa

Ejemplo 1: Diseñar, escribir y ejecutar un programa donde se pida el radio de un circulo, las unidades y calcule el area que deberá de sacar por pantalla.

Ejecutamos el programa

Ya tenemos el programa escrito. Ahora lo ejecutaremos. Si hay algún error en la escritura aparecerá en la ejecución.

Para ejecutarlo vamos a la pestaña de **Ventana de comandos** e introducimos el nombre sin la extensión .m.



Tecleamos el nombre del programa

Nos pide la info.

Nos da el resultado.

el programa está correcto!

5.- Diseñar un programa

GNU Octave

Ejemplo 2: Crear un programa que resuelva la ecuación de segundo grado:

$$ax^2 + bx + c = 0 \quad \Rightarrow \quad x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Definimos lo que tenemos que hacer:

¿qué parámetros necesitamos? ⇒ Vamos a necesitar que el usuario del programa nos diga los parámetros a, b, y c.

¿qué algoritmo tenemos que implementar? ⇒ Las dos soluciones dadas por la ecuación que define x

$$\begin{cases} x_1 = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a} \\ x_2 = \dfrac{-b - \sqrt{b^2 - 4ac}}{2a} \end{cases}$$

¿Cómo mostrar los resultados? ⇒ Vamos a mostrar los resultados por pantalla.

5.- Diseñar un programa

Ejemplo 2: Crear un programa que resuelva la ecuación de segundo grado:

$$ax^2 + bx + c = 0 \quad \Rightarrow \quad x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Escribimos el programa en el Editor:

ec_2_orden.m

```
% Programa solución ecuación de segundo grado
%
% Definición de variables. formato.
%
% Entrada de datos
%
disp('Resolucion ec. de segundo grado ax^2+bx+c=0')
%
a=input('El valor de a=');
b=input('El valor de b=');
c=input('El valor de c=');
%
% algoritmo de calculo
%
x1=(-b+sqrt(b^2-(4*a*c)))/(2*a)
x2=(-b-sqrt(b^2-(4*a*c)))/(2*a)
%
% muestra el resultado.
%
fprintf('Las raíces son: %.2f y %.2f\n',x1,x2);
```

Lo ejecutamos en la ventana de comandos

Ventana de comandos

```
>> ec_2_orden
Resolucion ec. de segundo grado ax^2+bx+c=0
El valor de a=1
El valor de b=2
El valor de c=-1
x1 = 0.41
x2 = -2.41
Las raíces son: 0.41 y -2.41
>>
```

Probamos que nos salga el resultado correcto.

5.- Diseñar un programa

Ejemplo 2: Crear un programa que resuelva la ecuación de segundo grado:

$$ax^2 + bx + c = 0 \quad \Rightarrow \quad x = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Pruebas:

Comprobamos que sale bien todo.

```
>> ec_2_orden
Resolucion ec. de segundo grado ax^2+bx+c=0
El valor de a=3
El valor de b=1
El valor de c=2
x1 = -0.17
x2 = -0.17
Las raíces son: -0.166667 y -0.166667
```

Hemos comprobado que para raíces reales (cuando $b^2 > 4ac$) el programa da el resultado correcto.

Pero para raíces complejas, que se obtienen cuando $b^2 < 4ac$ el programa no obtienen el resultado correcto. Solo representa el valor real.

Para obtener el resultado correcto necesitamos otras herramientas como sentencias del tipo **if** que veremos en la sección de bucles.

Funciones

6.- Funciones

GNU Octave

Conceptos:

- Las funciones son programas por sí mismos e independientes del programa principal y se deben de guardar en un archivo **.m** independiente

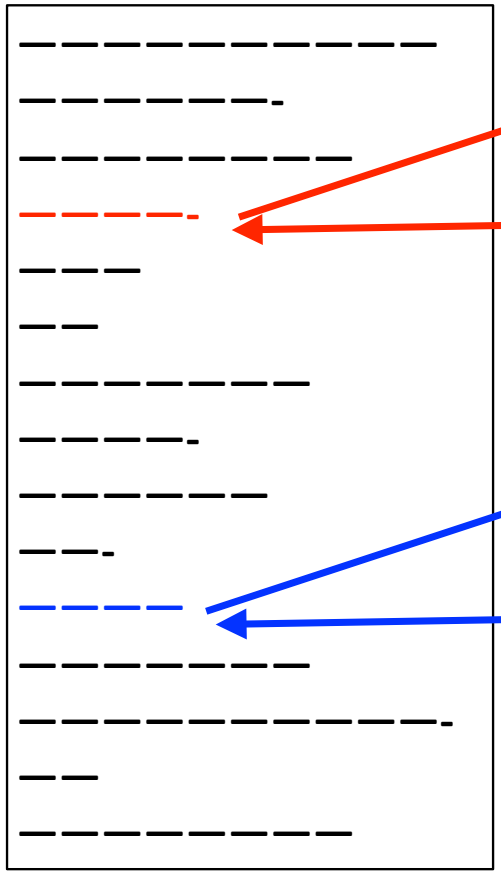
- Se tendrá un programa principal que llamará a las funciones y les pasará las variables de entrada que necesite.

- Las variables les pasará un resultado y en el programa principal definiremos una variable donde se almacene ese resultado.

- Las funciones actúan como “cajas negras”, lo que ocurra en una función se queda dentro de ella: Las variables que definimos en la función solo valen ahí, en el programa principal no existen como variables.

6.- Funciones

Idea de lo que es una función:



Llamada a función 1

Se ejecuta la función 1

Se dan operaciones dentro de la función 1

Devuelve el resultado antes de que se ejecute la siguiente línea del programa

Llamada a función 1

Se ejecuta la función 1

Se dan operaciones dentro de la función 2

Devuelve el resultado antes de que se ejecute la siguiente línea del programa

6.- Funciones

GNU Octave

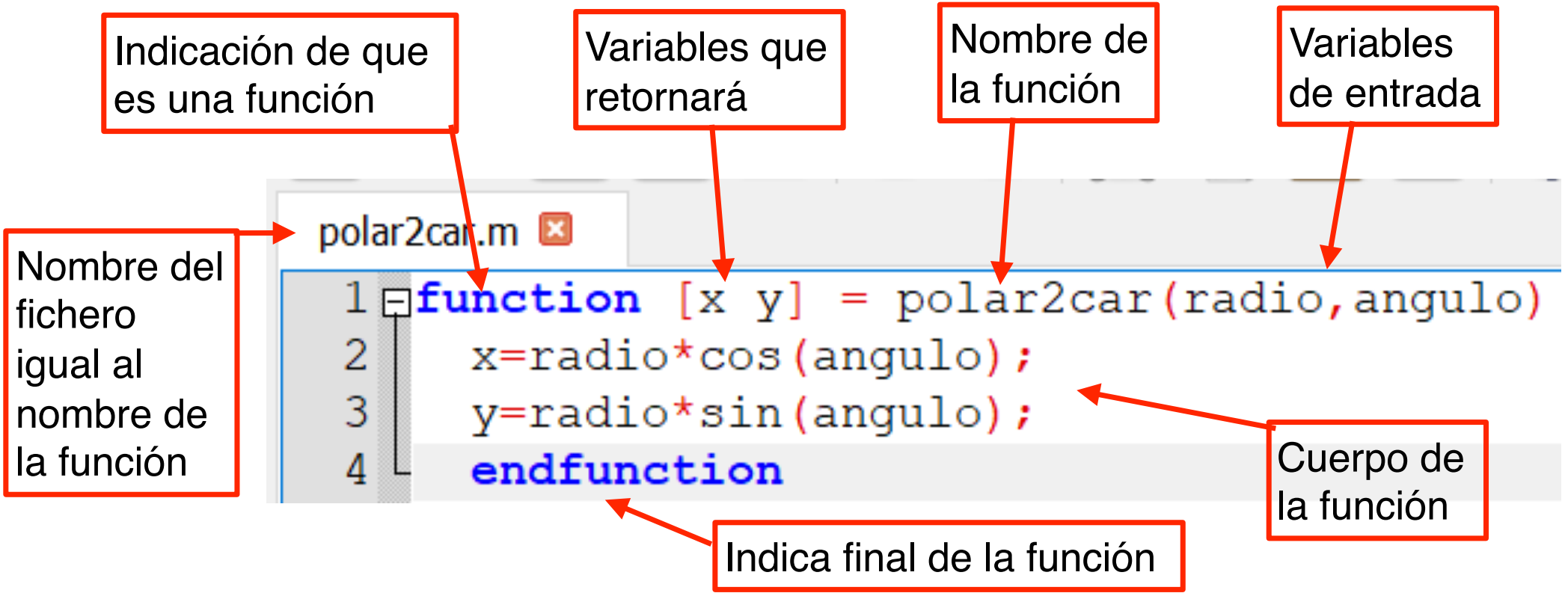
Formato:

- **function variable/s retornada/s = Nombre de la función (variables de entrada)**
- **Cuerpo de la función**
- **end**
  - Empiezan con la palabra clave **function** que indica el inicio de la función.
  - Variables retornadas, es opcional si queremos retornar cualquier dato al programa principal. Se puede definir una variable o una lista de variables entre corchetes [].
  - Si no existe las variables retornadas entonces el signo igual no tendrá que aparecer.
  - Nombre de la función a usar.
  - Opcionalmente siguen entre paréntesis la lista de argumentos. Si no hay la función no tendrá parámetros de entrada.
  - A continuación le sigue el Cuerpo de la función donde se implementan las operaciones o algoritmos que se consideren
  - La función termina con la palabra **end** o **endfunction**.

6.- Funciones

Formato:

Ejemplo: función que realiza una transformación de coordenadas polares a cartesianas:



6.- Funciones

Formato:

Llamada desde el programa principal a la función:

- **variable/s retornada/s = Nombre de la función (variables de salida)**

Ejemplo: función que realiza una transformación de coordenadas polares a cartesianas:

```
10  % algoritmo de calculo
11  %
12  [x,y]=polar2car(a,b);
```

Variables retornadas

Nombre de la función

Variables necesarias

*Vemos que las variables en el programa principal y en la función pueden llamarse de forma diferente. Esto es porque son independientes. Solo importa la posición.

6.- Funciones

Formato: Uso de una función con dos variables de entrada y dos de retorno

Ejemplo: función que realiza una transformación de coordenadas polares a cartesianas:

Programa principal:

polar2car.m | coordenadas.m

```
1  % Programa pasar coordenadas polares a cartesianas
2  %
3  % Definición de variables. formato.
4  %
5  % Entrada de datos
6  %
7  a=input('radio=');
8  b=input('angulo(en radianes)=');
9  %
10 % algoritmo de calculo
11 %
12 [x,y]=polar2car(a,b);
13 %
14 % muestra el resultado.
15 %
16 fprintf('Las coordenadas cartesianas son x=%f y=%f \n',x,y);
```

Llamada a función

Retorno del resultado

Función:

polar2car.m

```
1  function [x y] = polar2car(radio,angulo)
2    x=radio*cos(angulo);
3    y=radio*sin(angulo);
4    endfunction
```

Ejecuta la función

Ejecución en ventana de comandos

Ventana de comandos

```
>> coordenadas
radio=1
angulo(en radianes)=pi/6
Las coordenadas cartesianas son x=0.866025 y=0.500000
>>
```

6.- Funciones

Otros ejemplos: uso de una función con dos variables de entrada y una de retorno

Programa que suma dos números pedidos al usuario.

suma.m | funcionS.m

```
1  %programa suma
2  a=input('a=');
3  b=input('b=');
4  % realizar la suma
5  c=a+b
6  #c=funcionS(a,b);
7  fprintf('la suma es = %.2f\n',c);
8
```

Programa sin uso de una función

FunciónS.m suma x e y, devuelve una variable

suma.m | funcionS.m

```
1  %programa suma
2  a=input('a=');
3  b=input('b=');
4  % realizar la suma
5  #c=a+b
6  c=funcionS(a,b);
7  fprintf('la suma es = %.2f\n',c);
8
```

Programa con uso de una función

suma.m | funcionS.m

```
1  function S=funcionS(x,y)
2    S=x+y;
3    end
```

Externalizamos el cáculo!

El nombre de las variables puede ser diferente

6.- Funciones

Otros ejemplos: uso de una función sin retorno de variables ni variables de entrada

Programa que suma dos números pedidos al usuario.

suma.m | funcionS.m | sumaOK.m | funcionOK.m

```
1  %programa suma
2  a=input('a=');
3  b=input('b=');
4  % realizar la suma
5  #c=a+b
6  c=funcionS(a,b);
7  funcionOK;
8  fprintf('la suma es = %.2f\n',c);
```

Incluimos una función sin variables

suma.m | funcionS.m | sumaOK.m | funcionOK.m

```
1  function funcionOK
2    disp('La funcion esta OK')
3    endfunction
```

Nos dice simplemente que la función está OK.

Ejecución:

Ventana de comandos

```
>> sumaOK
a=2
b=3
La funcion esta OK
la suma es = 5.00
>>
```

6.- Funciones

Esquema general uso de m variables de retorno y n variables de entrada

Programa principal

```
[a,b,…,m]=nombrefun(ab,ac,..,n)
```

Función

```
function  [z,x…,m1]=nombrefun(a1,a2,a3,..,n3)
    Cuerpo de la funcion
endfunction
```

Recordad: no tienen porque llamarse iguales las variables que están en el programa principal y en la función. Simplemente, hay que respetar que haya el mismo número y seguir el orden.

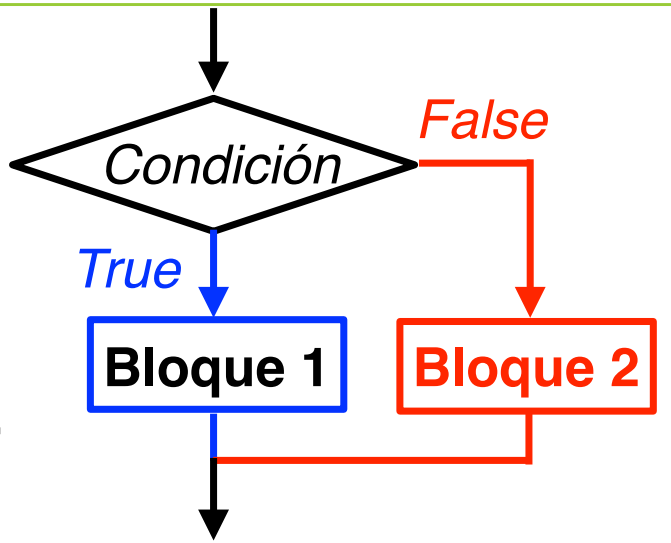
Bucles

6.- Bifurcaciones y Bucles

En todo lenguaje de programación podemos encontrar estas herramientas que son muy útiles para realizar programas:

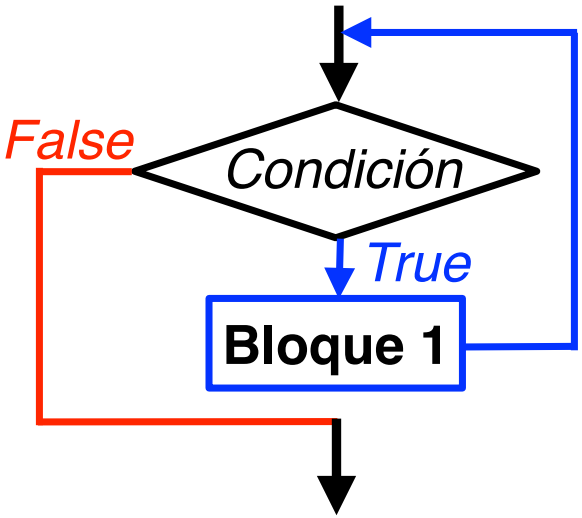
- **Bifurcaciones:** servirán para realizar una operación u otra dependiendo de la comparación con una variable.

→ **IF**



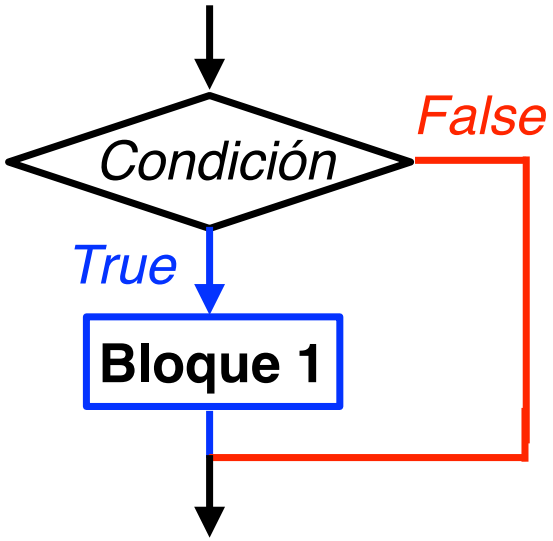
- **Bucles:** sirven para realizar varias iteraciones de una misma operación

→ **DO**
WHILE-UNTIL
FOR



GNU Octave

Sentencia **IF**



Se comprueba una condición y si es verdadera se ejecuta el código (bloque 1) dentro de la bifurcación. Si la condición es false no se hace nada:

Inicio de la bifurcación

Condición entre paréntesis: Con **operadores lógicos**

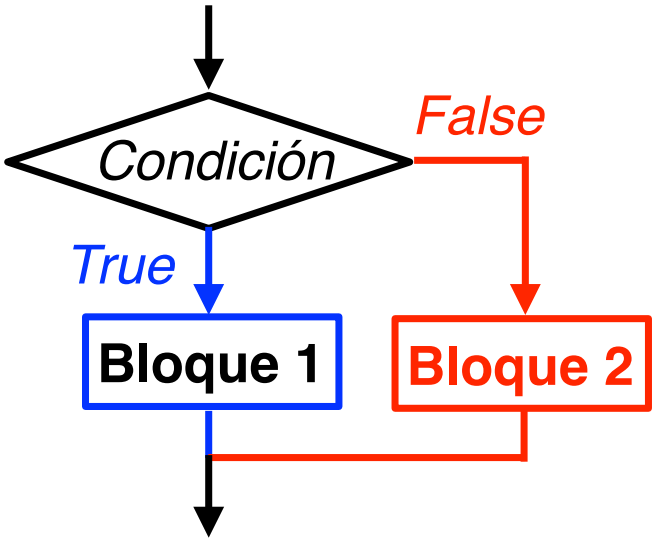
```
4 if (a>1);
5   A=a^2
6 endif;
```

Se ejecuta si a>1 es verdadero

Final de if

GNU Octave

Sentencia IF + **ELSE**



Else se usa junto a IF, si no se cumple la condición, se ejecuta el código que esta dentro de ELSE.

Inicio de la bifurcación

Condición entre paréntesis: Con operadores lógicos

```
4 if (a>1);
5   A=a^2
6 else
7   A=a+1
8 endif;
```

Bloque 1

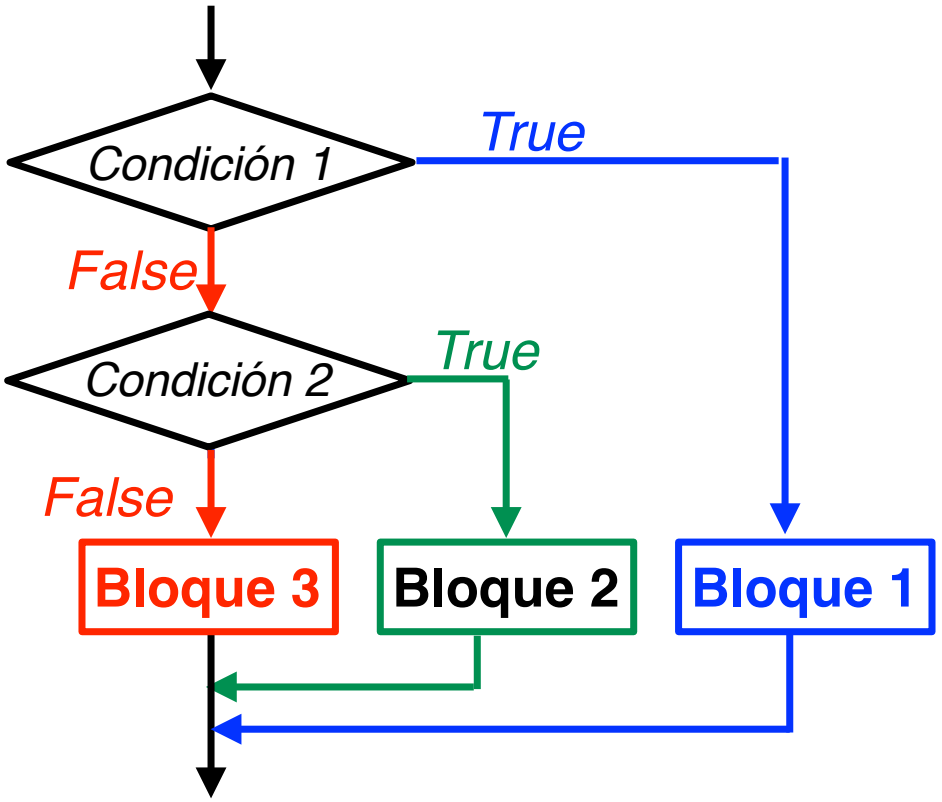
Bloque 2. Se ejecuta si la condición es falsa

Final de if

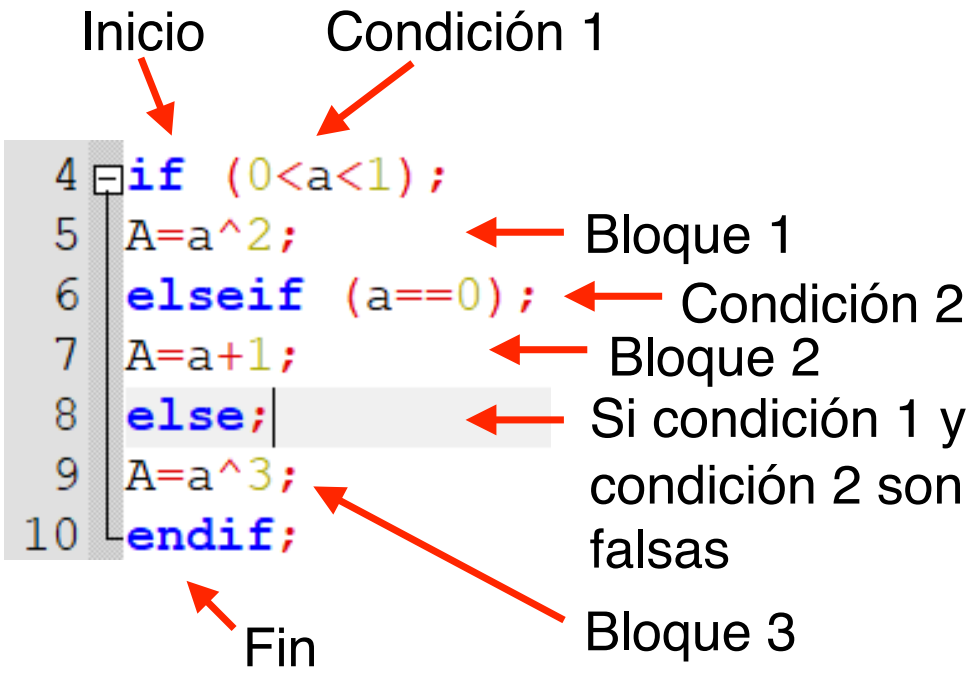
6.- Bifurcaciones y Bucles

GNU Octave

Sentencia IF + **ELSEIF** + ELSE



ELSEIF se usa junto a IF y ELSE para introducir otra condición si no se cumple la primera.

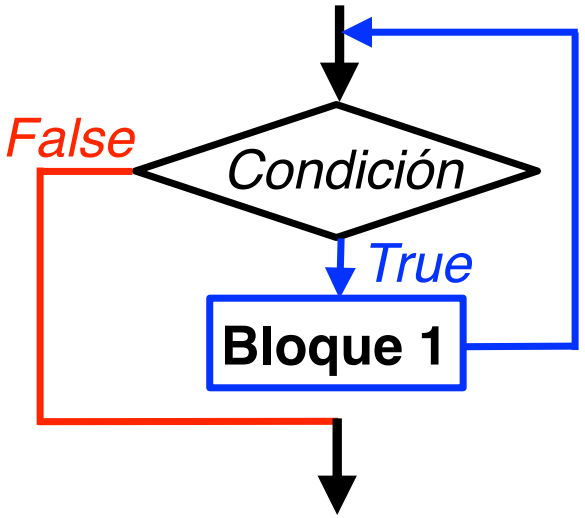


6.- Bifurcaciones y Bucles

GNU Octave

Sentencia **FOR**

Se ejecuta lo que haya dentro del FOR un número de veces. Sale del bucle cuando haya alcanzado las veces de repetición.



Inicio for

Condición: n=1 hasta 3

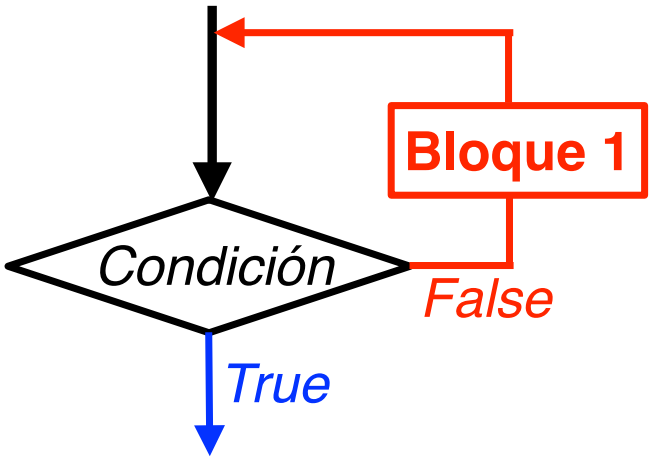
```
for n=1:1:3
    x(n)=a^2;
endfor
```

Bloque 1

Fin

GNU Octave

Sentencia **DO-UNTIL**



Se trata de repetir una serie de sentencias hasta que se cumple una condición. En el momento que la condición sea verdadera te saca del bucle. La condición se evalúa al final.

Inicio do

```
18  k=0;
19  do
20    y=a+k;
21    k++;
22  until(k==4)
23  disp(y);
```

Sentencias - bloque 1

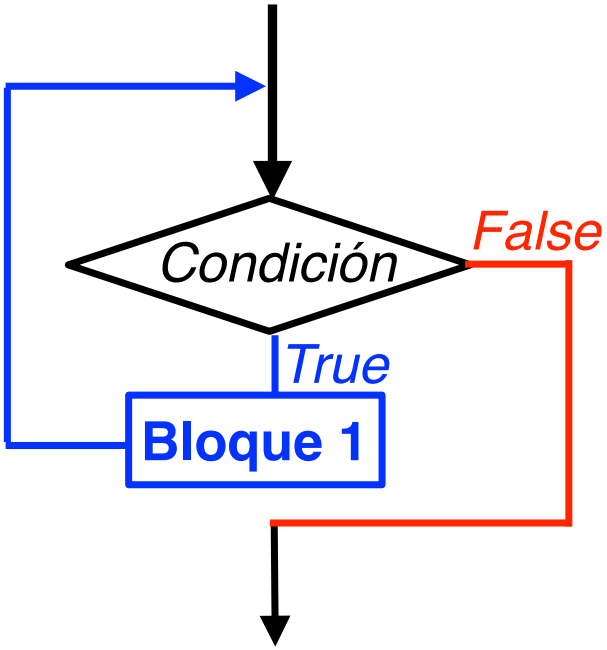
Condición

Finaliza cuando la condición se cumple

6.- Bifurcaciones y Bucles

GNU Octave

Sentencia **WHILE**



Se trata de repetir una serie de sentencias hasta que se cumple una condición. En el momento que la condición sea falsa el bucle se acaba. Se evalúa la condición al principio.

```
24  l=1
25  y=0
26  while (l<10)
27    y=a+l;
28    l++;
29  endwhile
30  disp(y)
```

Inicio while

Condición

Finaliza cuando la condición se cumple

Sentencias - bloque 1

Fin

6.- Bifurcaciones y Bucles

GNU Octave

Ejemplos: bucle if + función strcmp

strcmp(hola,'hola')

Sirve para comparar cadenas de texto. Si son iguales dará verdadero (1) y si no son iguales dará falso (0). El equivalente en número es ==

if_strcmp.m

```
1  %ejercicio comparar caracteres de texto
2  %
3  a=input('introduce texto: ','s');
4  if strcmp(a,'hola');
5  disp(a);
6  elseif strcmp(a,'adios');
7  disp(a);
8  else
9  disp('no coincide!');
10 endif
```

```
>> if_strcmp
introduce texto: aa
no coincide!
>> if_strcmp
introduce texto: ab
no coincide!
>> if_strcmp
introduce texto: hola
hola
```

6.- Bifurcaciones y Bucles

GNU Octave

Ejemplos: bucle do

bucle_do1.m

```
1  %comprobación que el usuario introduce un valor positivo
2  clear();
3  clc();
4  g=9.8
5  do
6    m=input('introduce un valor positivo de la masa =');
7  until m>0
8  p=m*g
9  fprintf('El peso es = %f N\n',p)
```

```
g = 9.8000
introduce un valor positivo de la masa =-1
introduce un valor positivo de la masa =-2
introduce un valor positivo de la masa =-3
introduce un valor positivo de la masa =-0.001
introduce un valor positivo de la masa =3
p = 29.400
El peso es = 29.400000
>>
```

6.- Bifurcaciones y Bucles

GNU Octave

Ejemplos: bucle for

bucle_do1.m | bucle_for1.m

```
1  %Ejemplo de for
2  % potencia
3  clear();
4  clc();
5  c=input('numero a elevar a potencia= ');
6  d=input('Introduce el exponente (entero)= ');
7  m=c;
8  for k=1:(d-1)
9    c=c*m;
10 endfor
11 fprintf('%d elevado a %d es %f \n',m,expo,c);
```

```
numero a elevar a potencia= 1.5
Introduce el exponente (entero)= 5
1.5 elevado a 5 es 7.593750
>>
```

6.- Bifurcaciones y Bucles

GNU Octave

Ejemplos: bucle for

bucle_do1.m | bucle_for1.m | bucle_while1.m

```
1  % While para comprobar usuario introduce valor positivo
2  clear();
3  clc();
4  g=9.8;
5  m=input('introduce valor de masa = ');
6  while m<=0
7    m=input('introduce un valor positivo de la masa= ');
8  endwhile
9  %
10 p=m*g;
11 fprintf('El peso es = %f N\n',p);
```

Ventana de comandos

```
introduce valor de masa = -1
introduce un valor positivo de la masa= 4
El peso es = 39.200000 N
>>
```

Lectura y Escritura de ficheros

6.- Lectura y escritura de ficheros

Muchas veces necesitamos guardar los resultados generados de un programa o abrir un fichero de texto en el que vienen datos necesarios. Para ello vamos a utilizar ciertas funciones integradas en el programa que nos permitirán hacerlo.

Antes vamos a ver comandos útiles para trabajar con ficheros, que se utilizarán en la ventana de comandos, estos admiten ser utilizados en forma de función:

type *nombre_fichero*: abre un fichero

pwd: muestra el directorio actual donde nos encontramos

dir: lista los archivos contenidos en el actual directorio.

cd: cambia de directorio. **cd ..** : cambia al directorio de más arriba

cd nombre_directorio: cambia al directorio *nombre_directorio*

6.- Lectura y escritura de ficheros

GNU Octave

Procedimiento habitual de lectura y escritura de ficheros:

```
11 % abrir archivo hola.txt guardado
12 % en variable file1
13 file1=fopen('hola.txt','w+')
14 %operaciones con el fichero
15 fprintf(file1,'%d',A)
16 %cierre del fichero
17 fclose(file1)
```

Abrir fichero

Operaciones escritura/lectura

Cerrar fichero

6.- Lectura y escritura de ficheros

GNU Octave

Apertura de ficheros

file1=**fopen**(‘filename’, permisos)

Abre el fichero *filename* con los permisos especificados y lo guarda como variable file1

Permisos

Van entre comillas también:
‘r’: abre el fichero para lectura
‘r+’ abre el fichero en modo lectura-escritura
‘w’ abre o crea un nuevo fichero en modo escritura. Si existe, se sobreescribe
‘w+’ abre o crea un fichero para lectura-escritura. Si existe, se sobreescribe.
‘a’ abre o crea un nuevo fichero para escritura. Si existe, añade al final.
‘a+’ abre o crea un nuevo fichero para lectura-escritura. Si existe, añade al final

ejemplo abrir fichero en modo lectura-escritura: **file1=fopen(‘hola.txt’,’r+’)**

6.- Lectura y escritura de ficheros

GNU Octave

Cerrar fichero

fclose('filename')

Cierra el fichero *filename.* Devuelve un cero si se cierra correctamente y un -1 si no.

fclose('all')

Cierra todos los ficheros. Devuelve un cero si se cierra correctamente y un -1 si no.

Escritura de datos en el fichero

Se usa la función **fprintf(filename,'datos a insertar',variables)**

```
fprintf(file1,'%d',A)
```

Escribe en el fichero *file1* la variable A en formato d (variable entera sin decimales)

6.- Lectura y escritura de ficheros

GNU Octave

Leer datos en un archivos

[variables]=fscanf(filename,formato, tamaño)

Esta función lee datos de un archivo llamado *filename* y los asigna a las variables con el formato especificado y el número de valores dados por el tamaño.

a=fscanf(file1,'%d', 1)

Nombre de la variable

Nombre Archivo que hemos identificado

Formato datos

Nº de datos a leer

Para leer matrices poner dimensiones entre []

6.- Lectura y escritura de ficheros

Ejemplo Realizar un programa que escriba un archivo y luego lo lea.

escritura.m

```
1  %programa escritura archivo
2  a=input('a=');
3  b=input('b=');
4  % abrir archivo hola.txt guardado
5  % en variable filel
6  filel=fopen('hola.txt','w+');
7  %operaciones con el fichero
8  k=0;
9  %
10 do;
11   A=a^k;
12   fprintf(filel,'%d %d\n',k,A);
13   k++;
14   until(b==k);
15   %cierre del fichero
16 fclose(filel);
```

```
17 % abrir el fichero solo lectura 'r'
18 filel=fopen('hola.txt','r');
19 % Almacenar en AA
20 AA=fscanf(filel,'%d %d',[2,k]);
21 %cerrar archivo
22 fclose(filel);S
23 %transponer el array para que coincida
24 %con los datos del archivo
25 AA=AA'
```

Resultado:

```
>> escritura
a=2
b=5
```

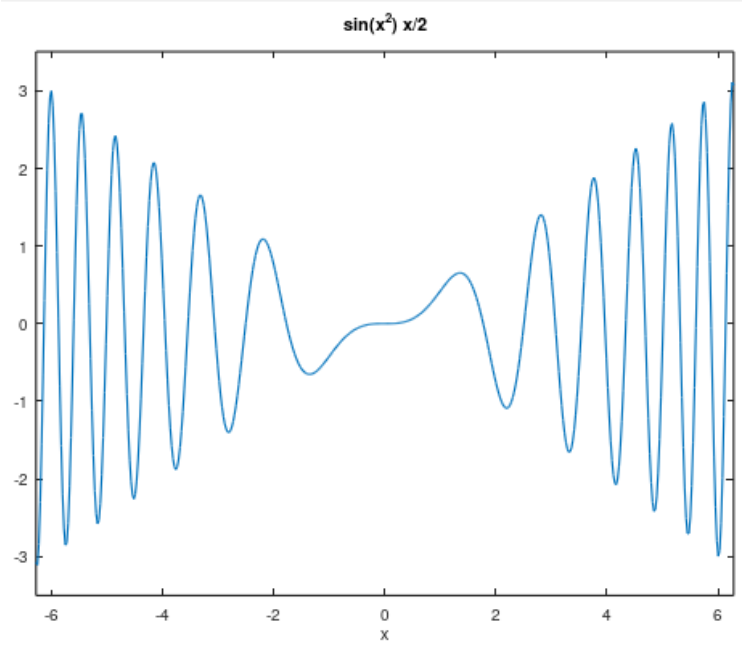
```
AA =

   0   1
   1   2
   2   4
   3   8
   4  16
```

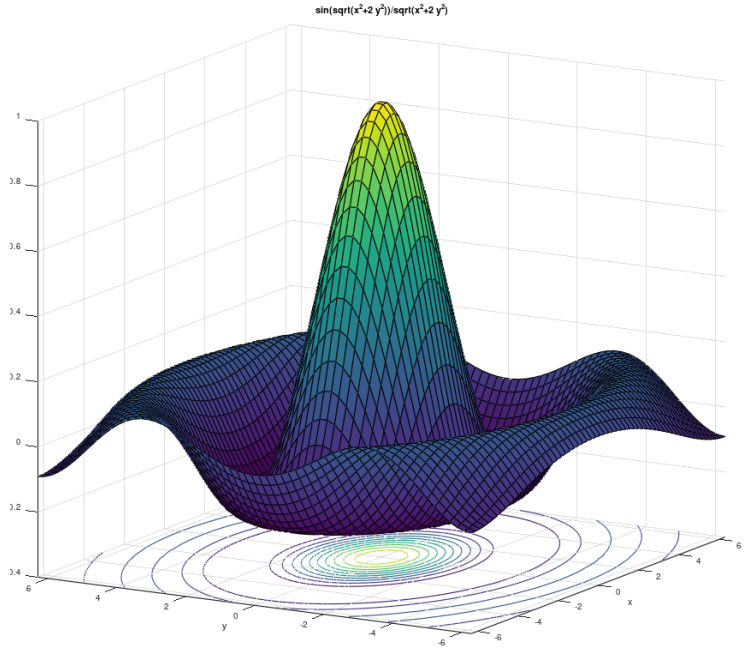
```
>> type hola.txt
0 1
1 2
2 4
3 8
4 16
```

Gráficos

8.- Gráficos

- Los gráficos en OCTAVE se generan en un ventana nueva con un programa llamado GNU plot.
- Están diseñados para la representación de matrices y vectores.
- Se pueden hacer desde gráficos 2D, 3D, de contorno, etc,..





8.- Gráficos

GNU Octave

Gráficos 2D

- **ezplot(‘función’,[rango])**

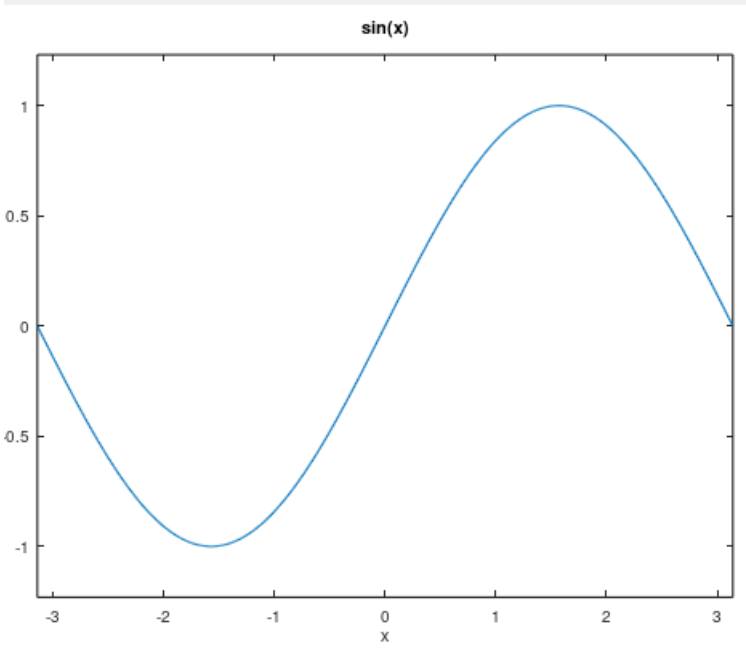
Con este comando podremos dibujar directamente funciones matemáticas. Formato:

Función

Rango en eje x

- **ezplot(‘sin(x/2)’,[-pi,pi])**

```
ezplot('sin(x/2)',[-pi,pi]);
```



8.- Gráficos

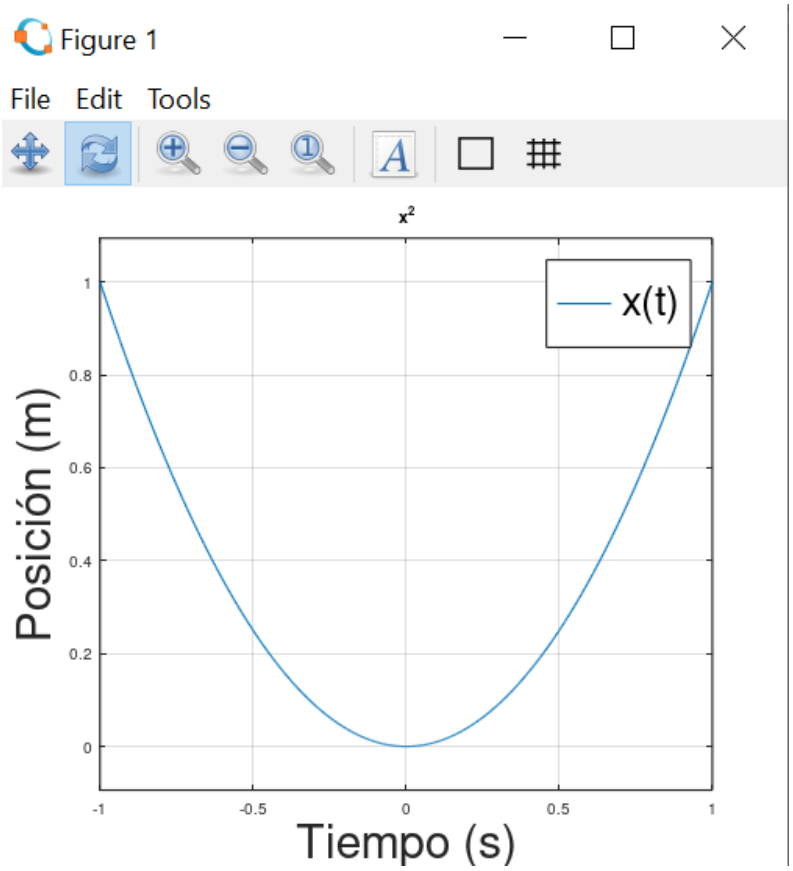
Gráficos 2D

- **ezplot('función',[rango])**

Opciones de gráfico, etiquetas ejes, leyendas

ex_ezplot.m

```
1  clf();
2  clear();
3  ezplot('x^2',[-1,1]);
4  xlabel('Tiempo (s)','FontSize',30);
5  ylabel('Posición (m)','FontSize',30);
6  legend('x(t)','FontSize',26);
```



8.- Gráficos

Gráficos 2D

- **plot(x,y,opciones)**

Genera curvas en 2D representando *y* frente a *x*.

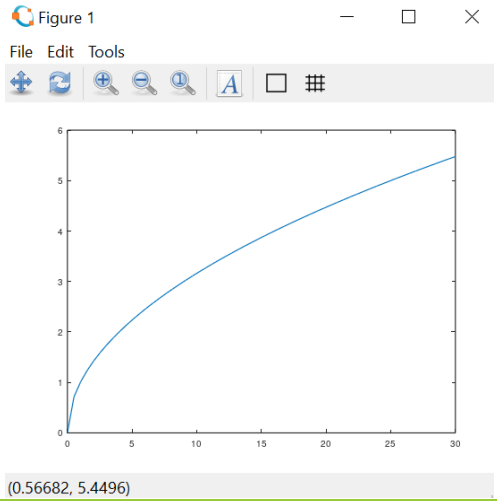
dibujo1.m

```
1  x=linspace(0,30,60);
2  y=sqrt(x);
3  plot(x,y);
```

Generar un vector x

Función

Representación

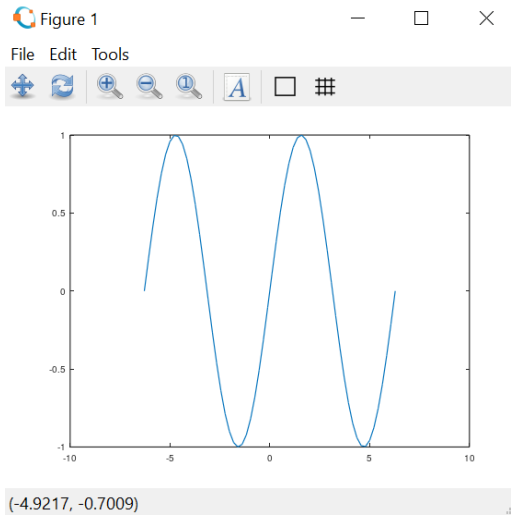


* color de las curvas incluyendo la siguiente opción **plot(x,y,'r')**

'r' rojo, 'y' amarillo, 'b' blue

dibujo1.m

```
1  x=linspace(-2*pi,2*pi,60);
2  y=sin(x);
3  plot(x,y);
```



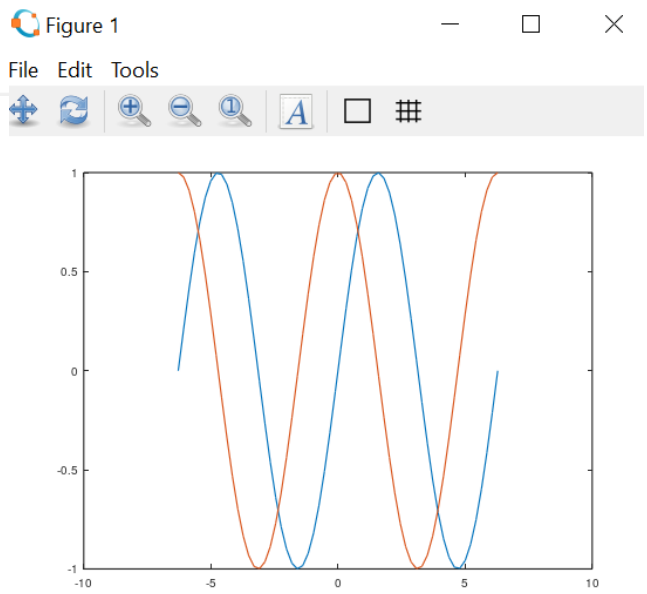
8.- Gráficos

Gráficos 2D

- **plot(x,y,opciones)**

dibujo1.m

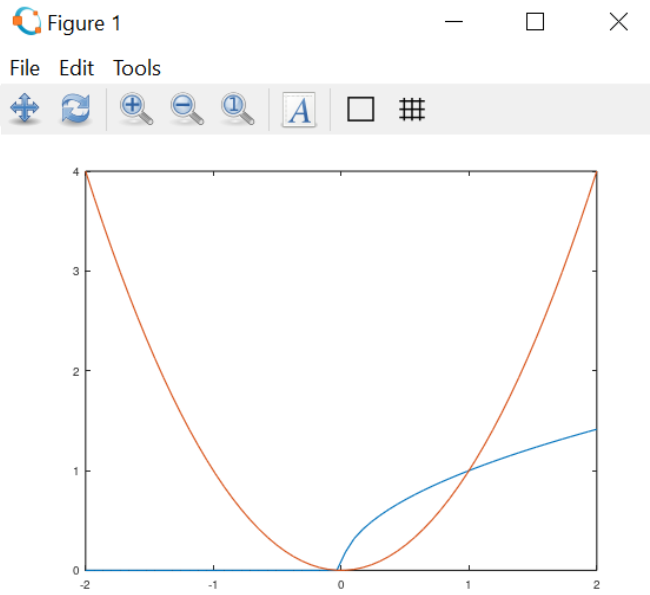
```
1  x=linspace(-2*pi,2*pi,60);
2  y=sin(x);
3  plot(x,y)
4  hold on;
5  y=cos(x);
6  plot(x,y);
```



Podemos dibujar más de una curva en una misma gráfica aplicando el comando **hold on**

dibujo1.m

```
1  x=linspace(-2,2,60);
2  y=sqrt(x);
3  plot(x,y);
4  hold on;
5  y=x.^2;
6  plot(x,y);
```



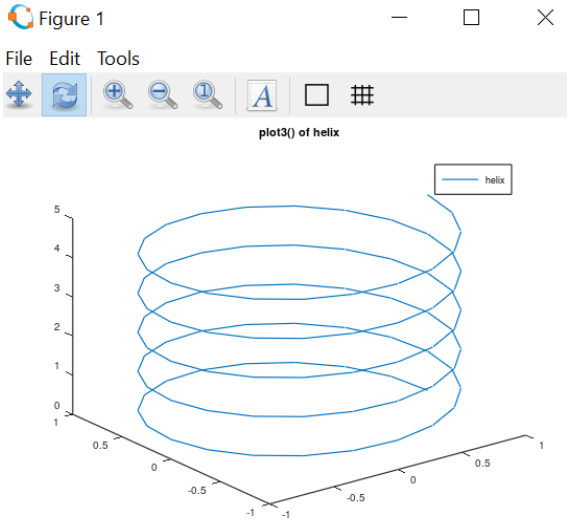
8.- Gráficos

Gráficos 3D

- **plot3(x,y,z)**

dibujo1.m | helix.m

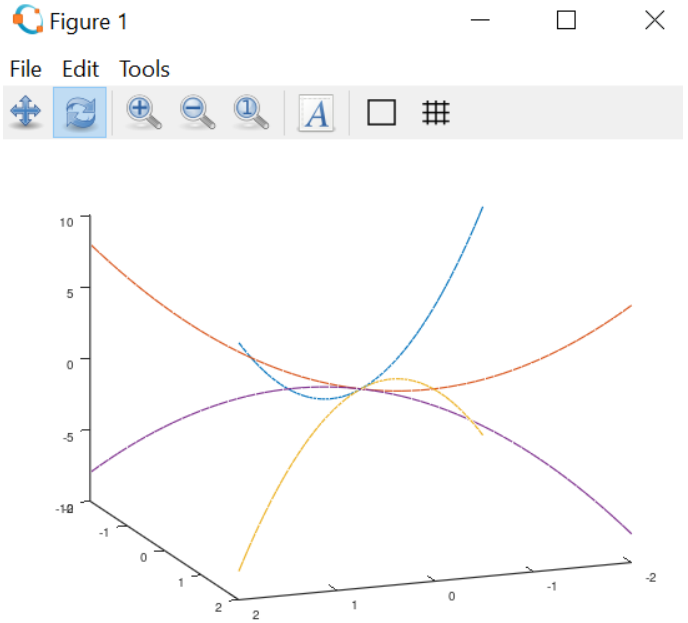
```
1 clf;
2 z=[0:0.05:5];
3 plot3(cos(2*pi*z),sin(2*pi*z),z);
4 legend("helix");
5 title("plot3() of helix");
```



Podemos dibujar una curva en 3D

dibujo1.m

```
1 x=linspace(-2,2,60);
2 y=linspace(-2,2,60);
3 z=x.^2+y.^2;
4 plot3(x,y,z);
5 hold on;
6 plot3(x,-y,z);
7 plot3(x,y,-z);
8 plot3(x,-y,-z);
```



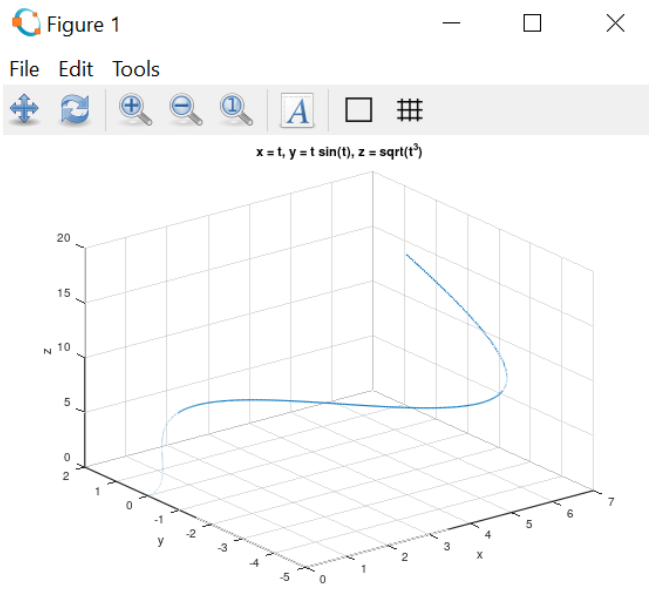
8.- Gráficos

Gráficos 3D

- **ezplot3(x,y,z)**

Podemos dibujar una curva paramétrica definida en 3D

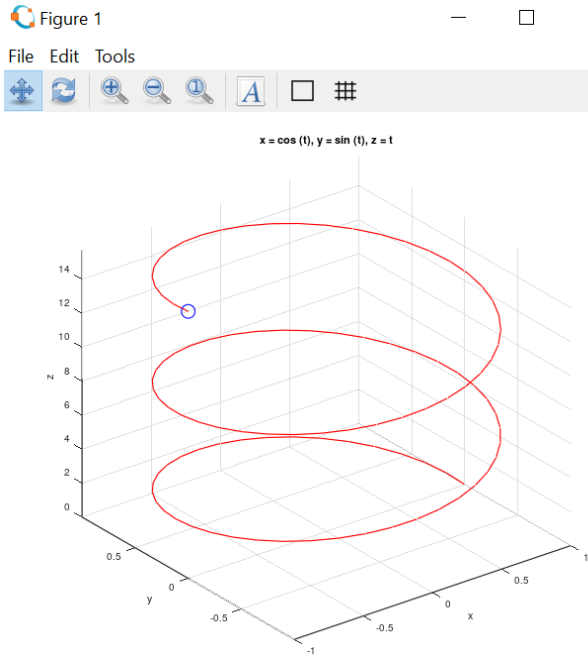
```
ezplot3('t','t*sin(t)','sqrt(t^3)')
```



dibujo1.m | helix.m | helix_parametrix.m

```
1 clf;
2 fx=@(t) cos(t);
3 fy=@(t) sin(t);
4 fz=@(t) t;
5 ezplot3(fx,fy,fz,[0,5*pi],100,"animate");
```

Animación:



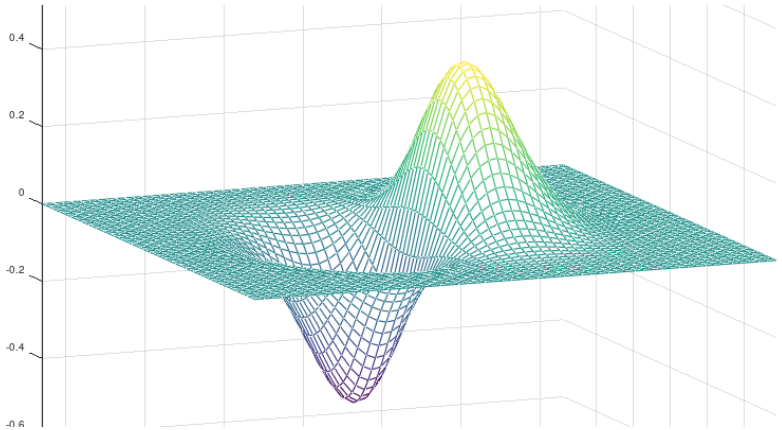
8.- Gráficos

GNU Octave

Gráficos 3D

- **ezmesh(x,y,z)**

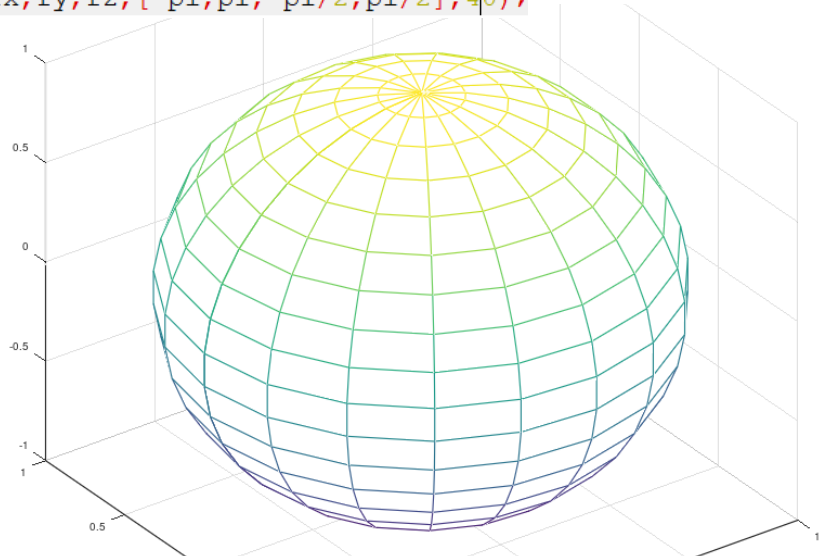
```
>> ezmesh('x*exp(-x^2-y^2)');
```



Podemos dibujar el mallado (mesh) de una superficie

ezmesh_ex1.m

```
1  clf;
2  colormap("default");
3  fx=@(s,t) cos(s).*cos(t);
4  fy=@(s,t) sin(s).*cos(t);
5  fz=@(s,t) sin(t);
6  ezmesh(fx,fy,fz,[-pi,pi,-pi/2,pi/2],40);
```



8.- Gráficos

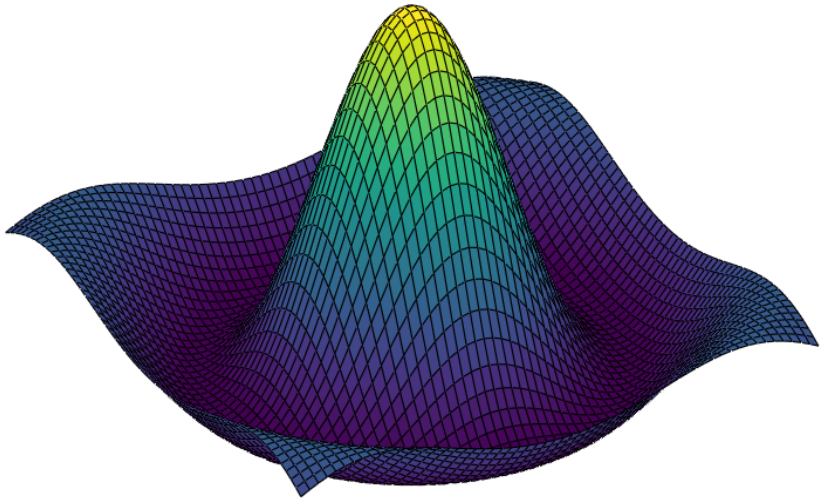
GNU Octave

Gráficos 3D

- **ezsurf('x y')**

Podemos dibujar la superficie definida por una función

```
ezsurf('sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)');
```



```
ezsurfc('x^2+y^2',"circ")
  colormap("ocean")
```