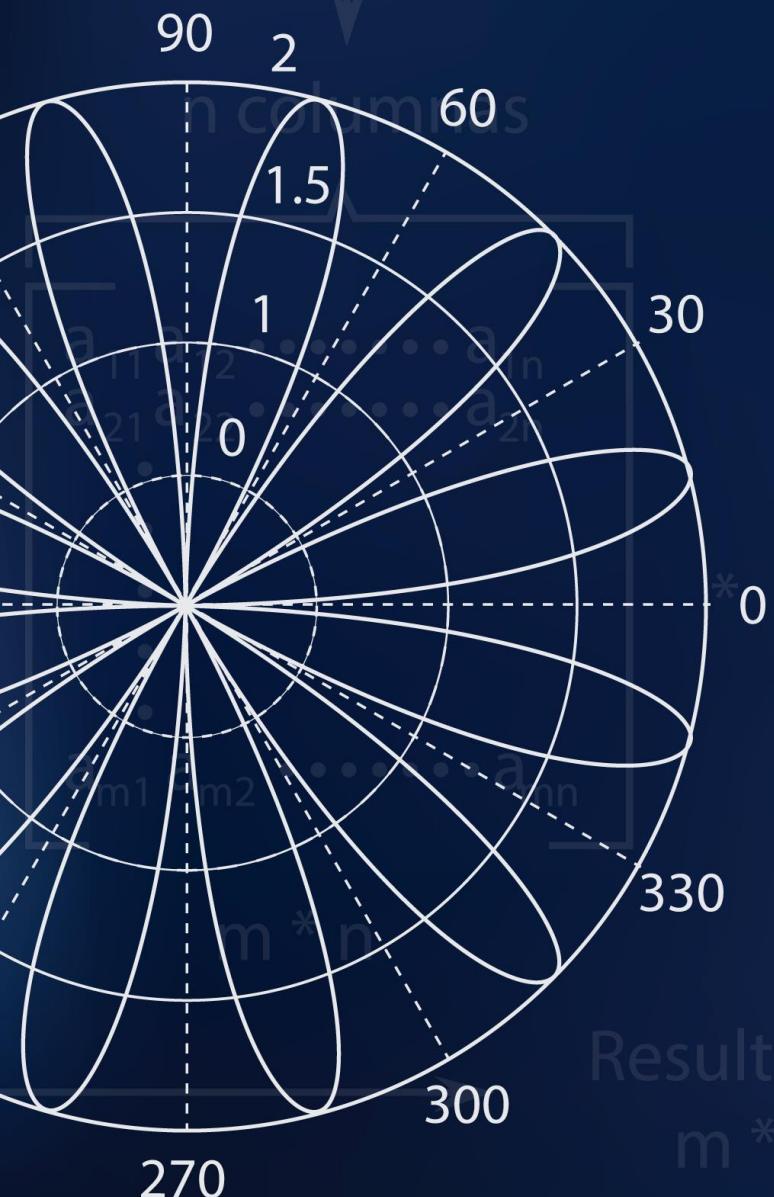
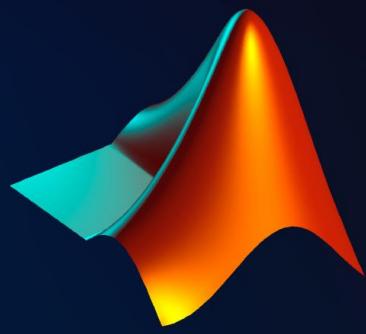
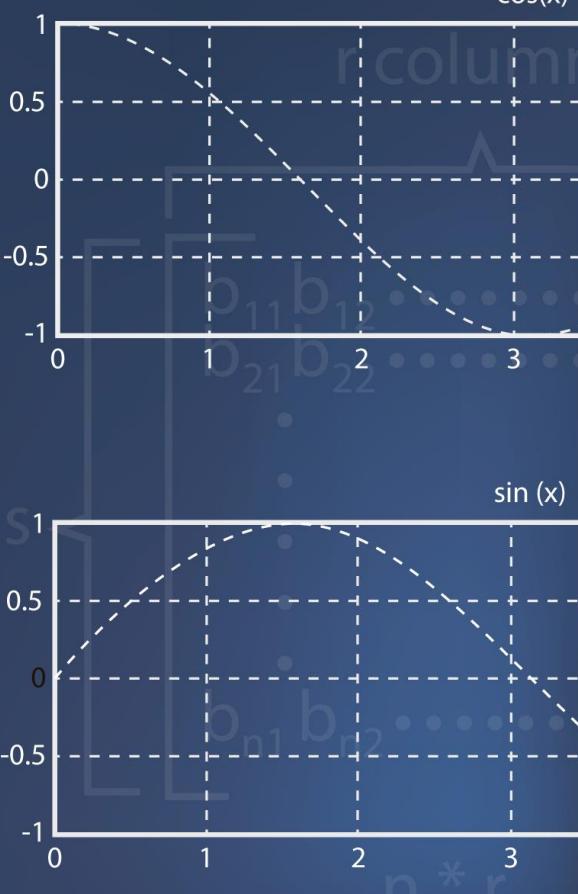


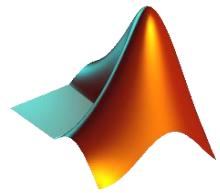
Curso MATLAB

Básico



Resultado:
 $m * r$





Contenido

Ambiente de Matlab	3
• Command window: Ventana de comandos.....	3
• Command History: Ventana del historial de comandos ...	5
• Current folder: Folder actual.....	6
• Workspace: Espacio de trabajo	8
Barra de herramientas y Navegador de ayuda.....	10
• Barra de herramientas y salir de Matlab.....	10
• Navegador de la Ayuda	11

Sesión 1: Entorno de Matlab

Ambiente de Matlab

- **Command window: Ventana de comandos**

La ventana de comandos es el principal mecanismo para trabajar con MATLAB. Las funciones introducidas, llamadas también "las entradas" se ejecutan pulsando la tecla Enter. Debemos tener en cuenta que al escribir los nombres de las funciones o de los comandos, MATLAB distingue entre mayúsculas y minúsculas (por lo general, las funciones se escriben en minúsculas).

A continuación una lista de comandos útiles en la línea de comandos:

clc: Borra el contenido de la pantalla y coloca el cursor en la primera línea.

Esc: Borra la línea.

%: Todo lo que aparece detrás del símbolo % y en la misma línea se considera un comentario.

```
>> %esto es un comentario
```

En, en: Notación científica. Ej.: E6, E-3, e4

```
>> 0.645e9  
ans =  
645000000
```

[ctrl]+[c]: Detiene la ejecución de cualquier comando o función.

Con las flechas del cursor ($\uparrow \downarrow$) se pueden recuperar las órdenes anteriores, sin tener que volver a escribirlas. La flechas ($\leftarrow \rightarrow$) permiten presentar el desplazamiento horizontal en la línea de comandos. Estas flechas son de mucha utilidad en el caso de una equivocación o cuando queremos volver a ejecutar un mismo comando o hacerle una pequeña modificación.

A veces también es muy importante, que el resultado de un cálculo no aparezca en pantalla. Por ejemplo, si generamos una matriz de orden muy alto con el objeto de hacer después una gráfica. El hecho de que aparezca la matriz en pantalla puede resultar un poco engorroso. Para evitar mostrarlo en la pantalla ponemos un punto y coma (;) al final de la instrucción.

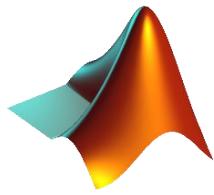
```
>> x=sin(50);           ← No muestra valor.
>> x=sin(50)

x =                   ← Muestra valor.

-0.2624
```

Los comandos se pueden ir escribiendo y ejecutando uno a uno, es decir, renglón a renglón, y también podemos escribirlo uno a continuación de otro en una misma línea, para lo cual se necesita que vayan separados por comas (,).

Si el comando o la cantidad de comandos son demasiado larga para que aparezca en un único renglón, se puede romper la cadena y seguir en el siguiente renglón, escribiendo tres puntos suspensivos (...+[ENTER]).



```
>> x=sin(50), y=cos(50), ...
z=tan(50)

x =
-0.2624

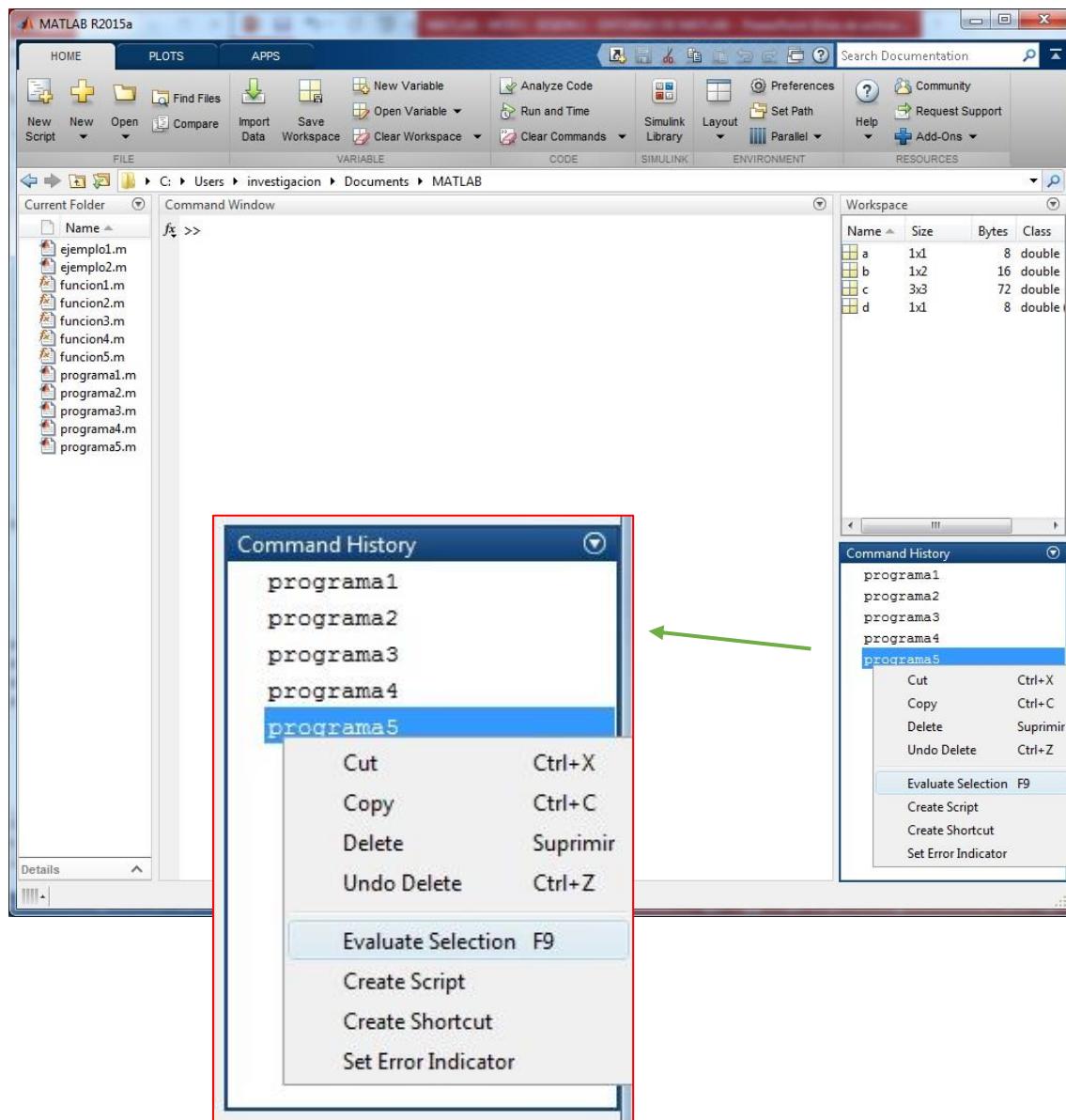
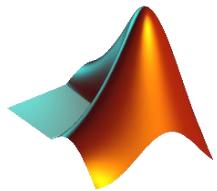
y =
0.9650

z =
-0.2719
```

- **Command History: Ventana del historial de comandos**

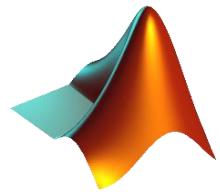
Esta ventana muestra una relación con las distintas funciones que han sido empleadas en la ventana de comandos.

Una vez seleccionadas una o varias líneas en esta ventana (al igual que en la ventana de comandos) el botón derecho del ratón permite su evaluación entre otras opciones (se muestra en la imagen siguiente). También podemos evaluarla haciendo doble clic sobre esta opción.

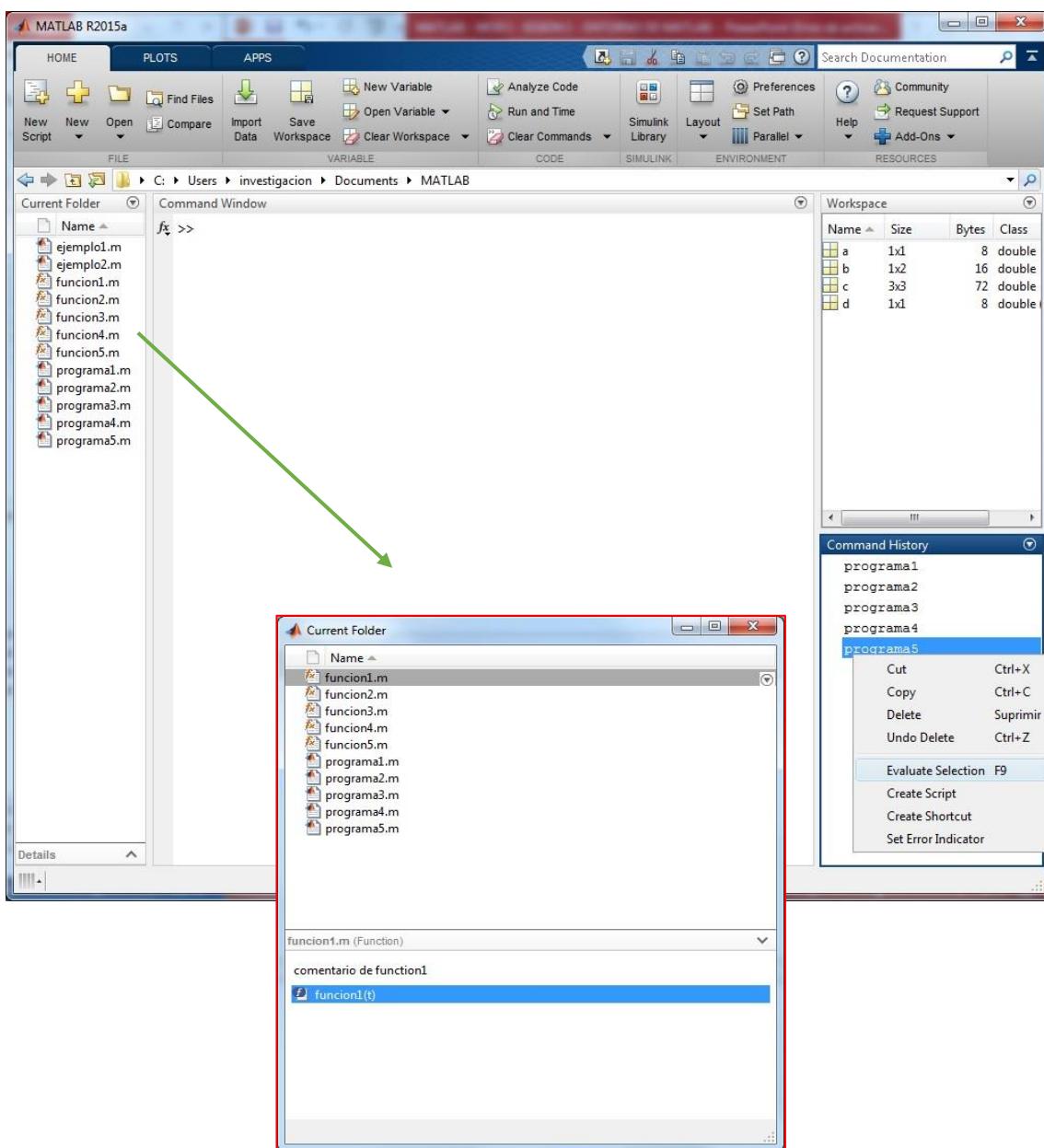


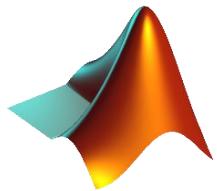
- **Current folder: Folder actual**

La ventana folder actual se sitúa en la parte izquierda del escritorio de MATLAB. Esta ventana se divide en 2 partes, Name y details. En la primera se muestra los ficheros del entorno de MATLAB que podemos abrir y hacer cambios y en la segunda se muestra los detalles que pueden tener estos ficheros como por ejemplo si son function o script además de los comentarios que puedan presentar. Para



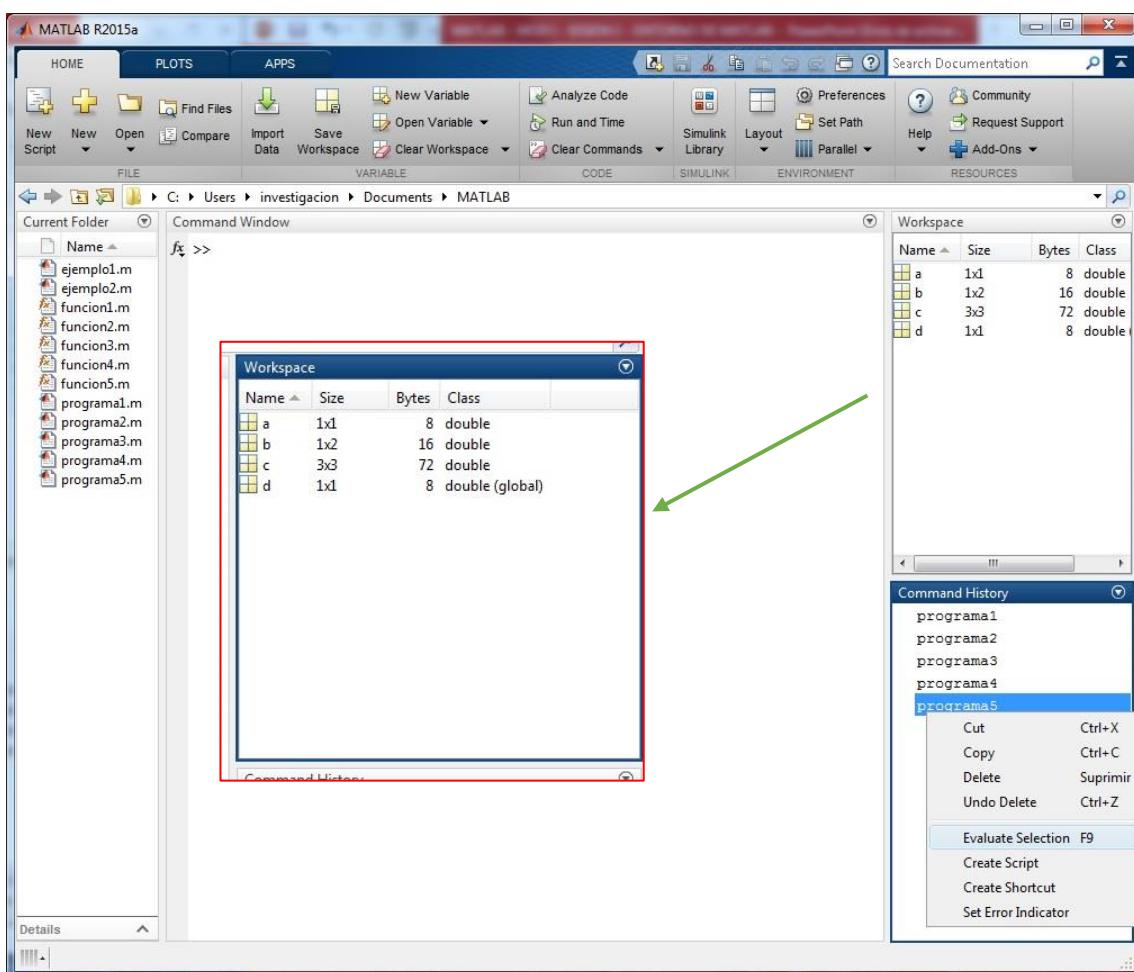
mostrar esta ventana separada del escritorio de MATLAB basta hacer clic en la opción undock del botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio de Matlab se utiliza la opción Dock del menú despegable en la esquina superior derecha de esta ventana.



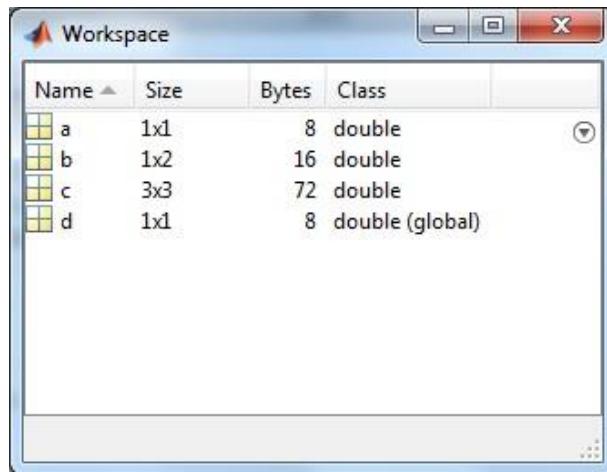
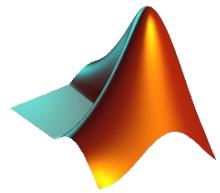


• **Workspace: Espacio de trabajo**

La ventana de espacio de trabajo (Workspace) se sitúa en la parte superior derecha del escritorio de MATLAB. Su función es ver las variables almacenadas en la memoria. Para cada variable se muestra su nombre, tipo, tamaño y clase, tal y como se indica en la figura de abajo.

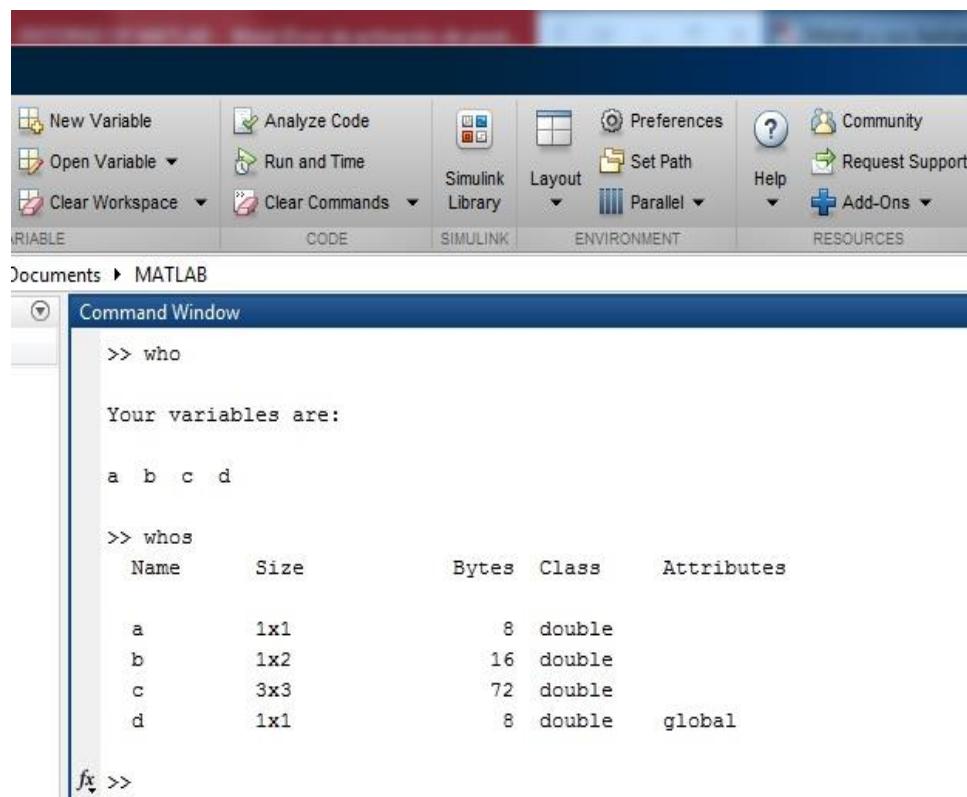


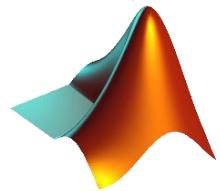
Para mostrar esta ventana separada del escritorio de MATLAB basta hacer clic en la opción undock del botón situado en su esquina superior derecha. Para retornar la ventana a su sitio en el escritorio se utiliza la opción Dock del mismo sitio.



Si queremos información acerca de las variables que estamos utilizando en Matlab podemos verlas en la ventana Workspace (espacio de trabajo) o usar:

- **who** para obtener la lista de las variables (no de sus valores).
- **whos** para obtener la lista de las variables e información del tamaño, tipo y atributos (tampoco da valores).



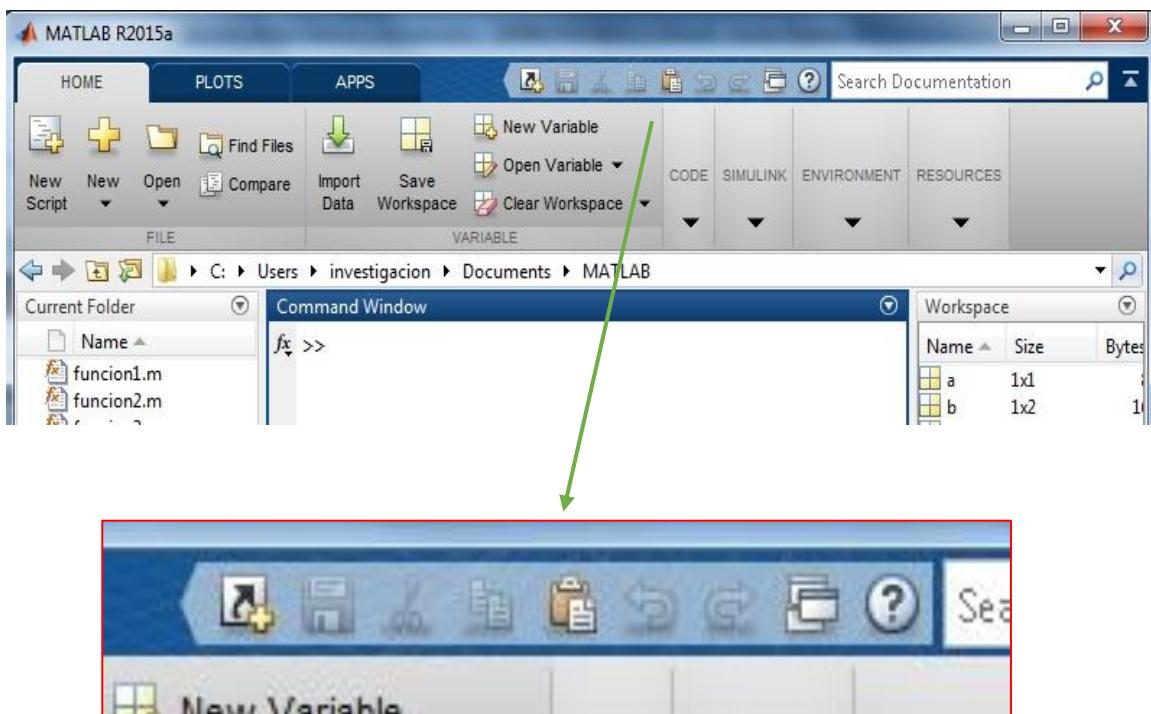


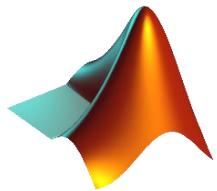
Barra de herramientas y Navegador de ayuda

- **Barra de herramientas y salir de Matlab**

La barra de herramientas de Matlab se encuentra por encima de las ventanas antes estudiadas, y en su parte superior reúne las opciones más comunes encontrando las siguientes opciones:

- New shortcut (Nuevo acceso directo).
- Save (Guardar).
- Cut (Cortar).
- Copy (Copiar).
- Paste (Pegar).
- Undo (Deshacer).
- Redo (Rehacer).
- Switch Windows (Cambiar de ventana).
- Help (Ayuda).



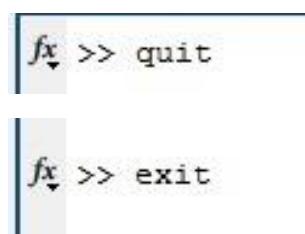


Y más abajo se encuentran opciones que se pueden realizar dependiendo del punto en el que nos encontremos con los cuales podemos crear un nuevo script, función, figura, abrir archivos recientes, buscar archivos, importar datos, guardar o limpiar los datos que tengamos en workspace, crear o abrir una variable.

Opciones con los códigos que estemos trabajando, Simulink, realizar distinta modificaciones en el ambiente de Matlab que nos permitan trabajar de forma más cómoda y los recursos que nos brinda Matlab como documentación y ejemplos de las funciones que nos brinda entre otros.

Para salir de Matlab podemos usar cualquiera de las siguientes formas:

- Escribir **quit** o **exit** en la ventana de comandos.
- Presionar **ctrl+q** ó el botón **x** (esquina superior derecha).

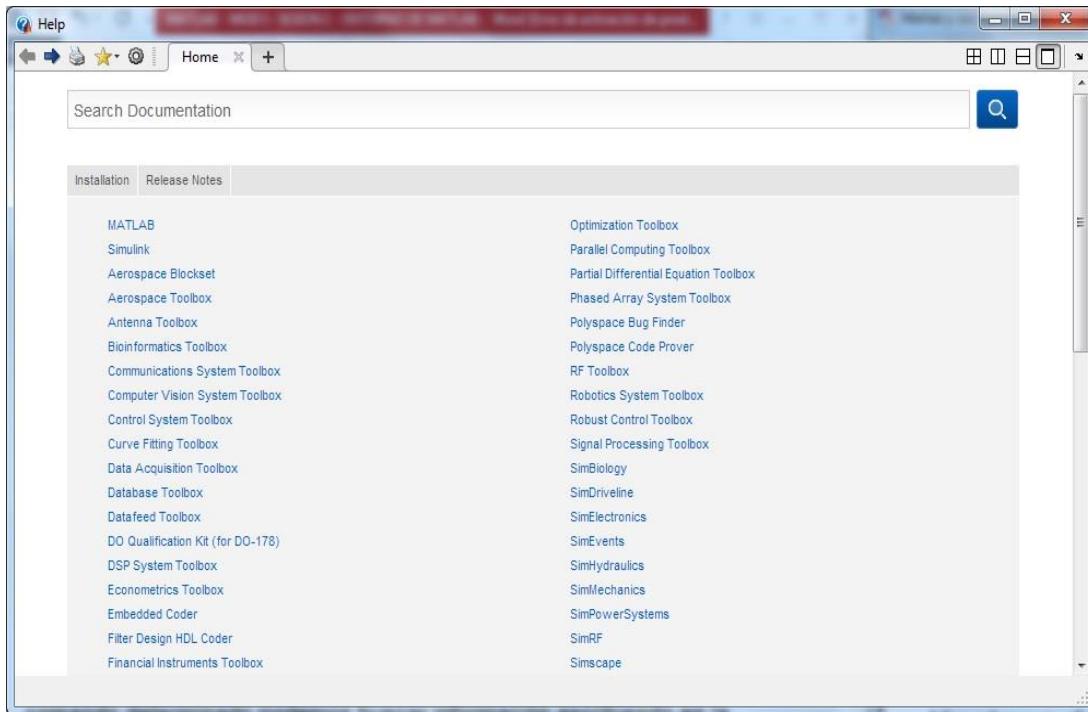
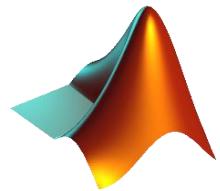


```
fx >> quit
```

```
fx >> exit
```

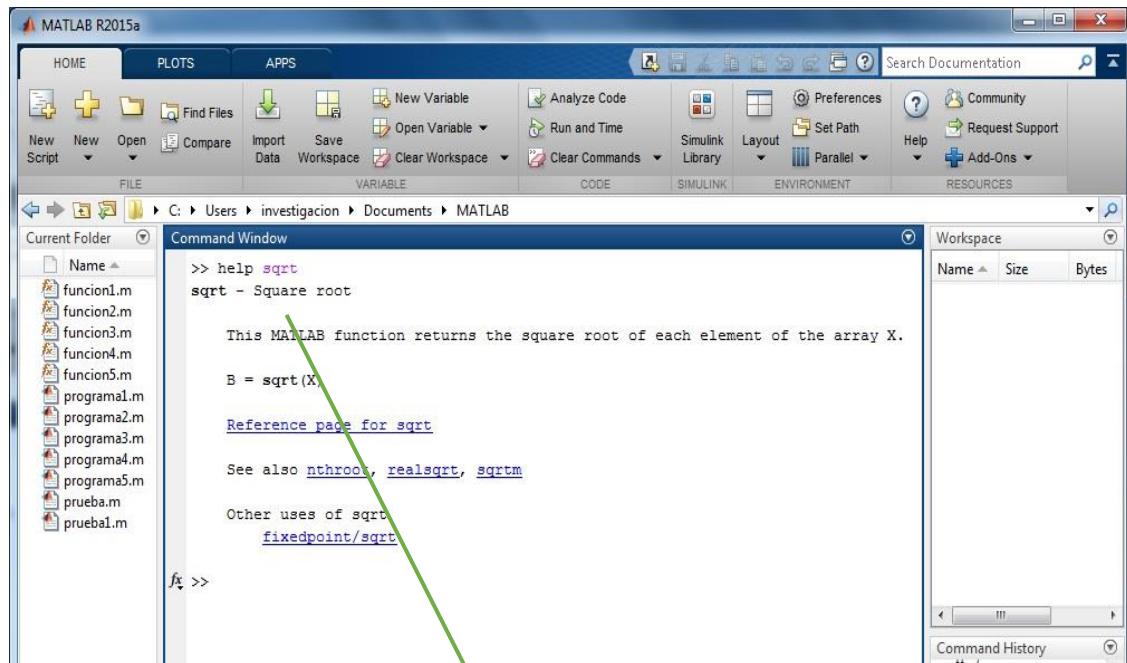
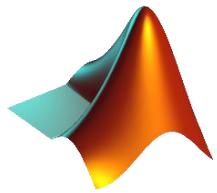
• Navegador de la Ayuda

Matlab proporciona asistencia de varios modos. El navegador de ayuda de Matlab se obtiene haciendo clic en el botón **?** de la barra de herramientas o utilizando la función **doc** (en versiones anteriores se usa **helpbrowser** pero esta función dejara de estar disponible en próximas versiones) en la ventana de comandos.



Aquí podemos navegar por todo el menú que nos brinda, como vemos Matlab nos permite desarrollar muchísimas áreas, para los cuales se tendría que mostrar en cursos especializados.

Si queremos consultar un comando determinado podemos buscar información escribiendo en la ventana de comandos **help <comando a consultar>**, o simplemente **help**. También podemos abrir la ventana de ayuda con el ratón o con la tecla **F1**. Una vez abierta esta ventana podemos buscar por contenidos, palabras concretas, y otros.



>> help sqrt
sqrt - Square root

This MATLAB function returns the square root of each element of the array X.

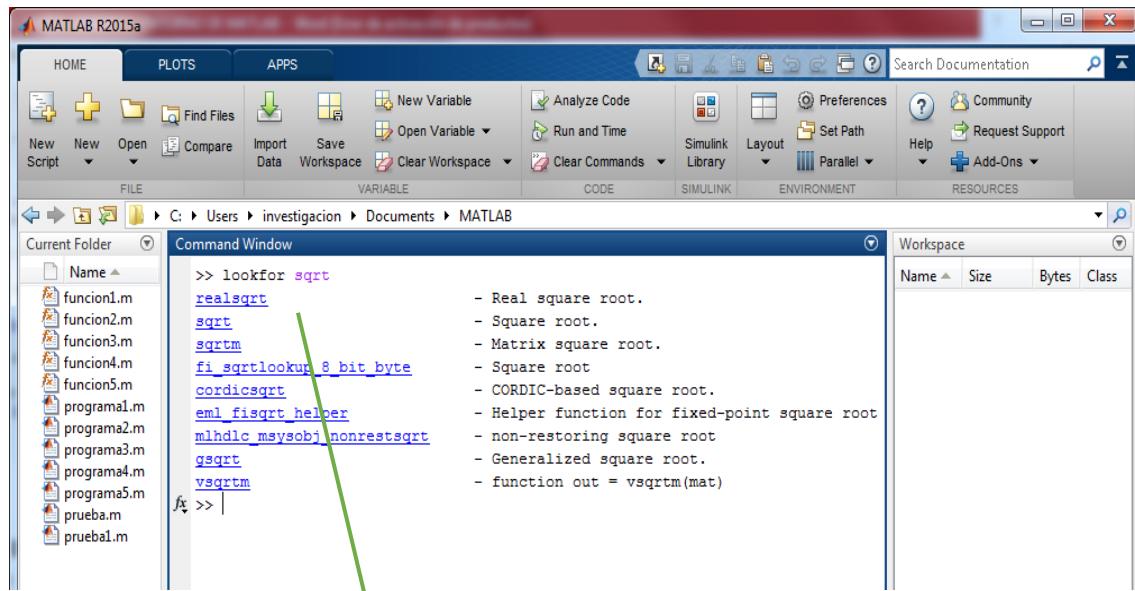
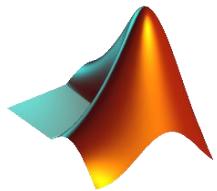
B = sqrt(X)

[Reference page for sqrt](#)

[See also nthroot, realsqrt, sqrtm](#)

[Other uses of sqrt](#)
[fixedpoint/sqrt](#)

Por último con la orden **lookfor <palabra>**, se busca en todas las primeras líneas de las ayudas de los temas de Matlab y devuelve aquellos que contienen la palabra clave que hemos escrito. No es necesario que la palabra clave sea una orden de Matlab.

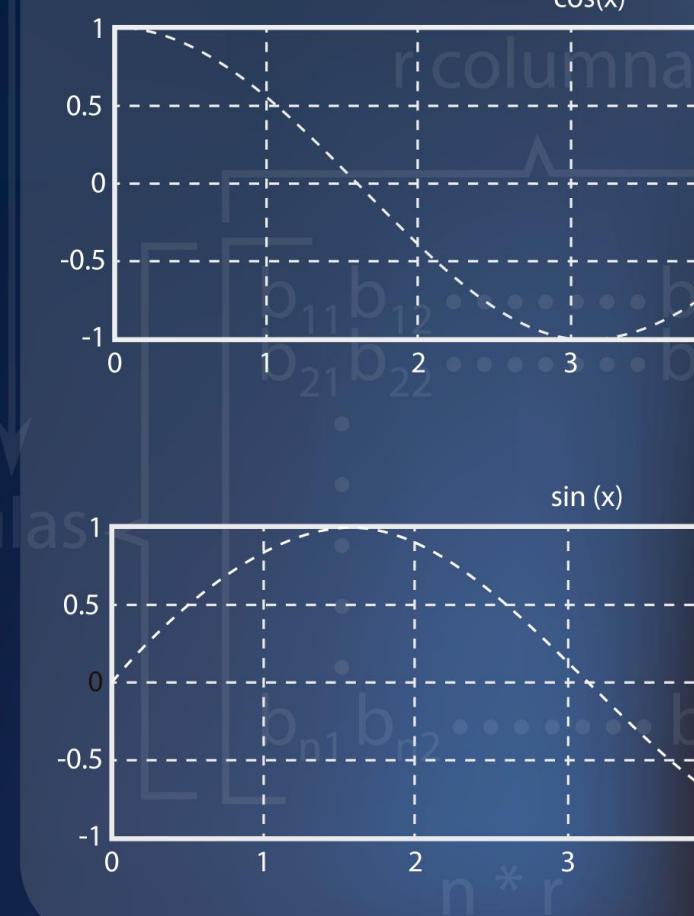
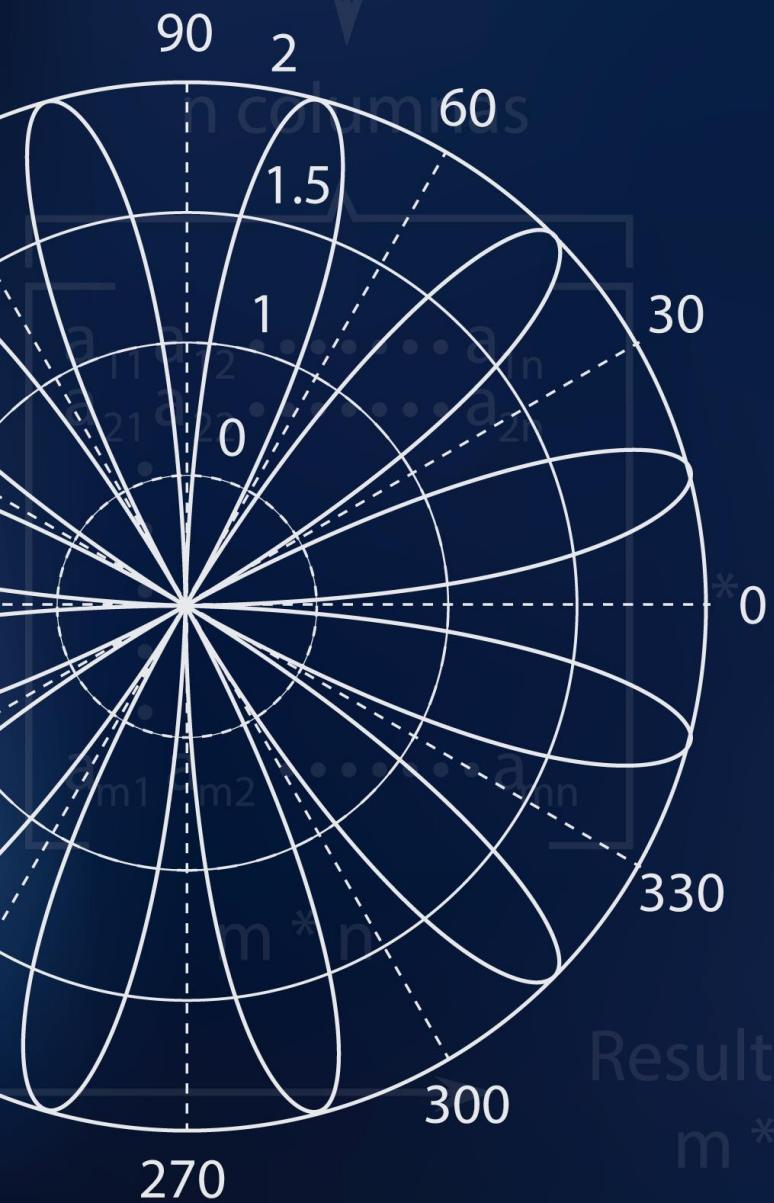
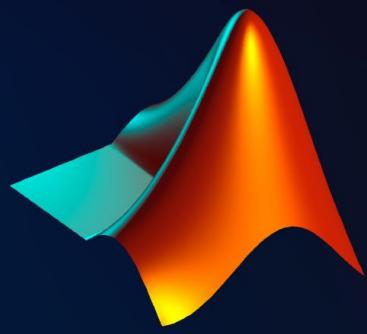


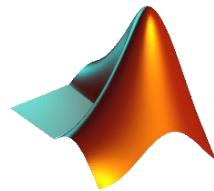
```
>> lookfor sqrt
realsqrt
sqrt
sqrtn
fi_sqrtlookup_8_bit_byte
cordicsqrt
eml_fisqrt_helper
mlhdlc_msysobj_nonrestsqrt
gsqrt
vsqrtm
fx >>
```

The same list of functions and their descriptions is shown, but the entire list is now enclosed in a red rectangular box, indicating the specific search results for the `sqrt` function.

Curso MATLAB

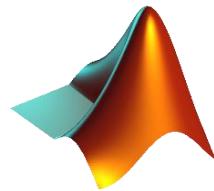
Básico





Contenido

Variable	3
Números enteros	6
Funciones con números enteros y divisibilidad	8
Operaciones en \mathbb{Q}	9
Funciones con argumento real	14
• Funciones trigonométricas	14
• Funciones hiperbólicas	15
• Funciones exponenciales y logarítmicas	16
• Funciones específicas de una variable numérica	16



Sesión 2: Números Reales

Variable

MATLAB no requiere ningún tipo de comando para declarar variables. Sencillamente crea la variable mediante asignación directa de su valor. Por ejemplo:

Command Window

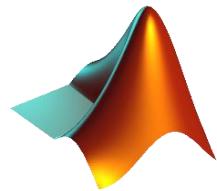
```
>> v=5
v =
5
```

La variable *v* valdrá 5 mientras no se cambie su valor mediante una nueva asignación. Una vez declarada la variable podemos utilizarla en cálculos.

```
>> v^2
ans =
25
>> v+30
ans =
35
```

Above the code block, there is a red box containing the text '^ : [Alt] + 94' with a blue arrow pointing from the '^' character in the first line of code to the box.

El valor asignado a una variable es permanente, hasta que no se cambie de forma expresa o hasta que no se salga de la presente sesión de MATLAB.



Si ahora escribimos:

```
>> v=5+4  
  
v =  
  
9
```

La variable *v* pasa a valer 9 a partir de este momento, tal y como se observa en el cálculo siguiente:

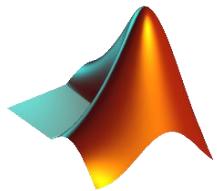
```
>> v^2  
  
ans =  
  
81
```

Los nombres de las variables comienzan por una letra seguida de cualquier número de letras, dígitos o subrayados. También es muy importante señalar que MATLAB es sensible a mayúsculas y minúsculas; por lo tanto, una variable con mayúsculas es distinta a la misma variable con minúsculas.

En Matlab se puede trabajar con diferentes tipos de números y expresiones numéricas, que abarcan todo el campo de los números enteros, racionales, reales y los números complejos, y se utilizan en argumentos de funciones.

Las operaciones aritméticas aquí se definen de acuerdo con las convenciones matemáticas estándar.

Matlab es un programa interactivo que permite realizar de manera sencilla gran variedad de operaciones matemáticas. Debido a que asume las operaciones aritméticas habituales de suma, diferencia, producto, división y potencia, con la jerarquía habitual entre ellas:



Símbolo	Operación
x+y	Suma
x-y	Diferencia
x*y	Producto
x/y	División
x^y	Potencia

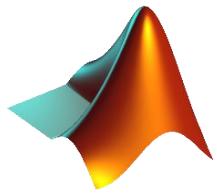
Para sumar dos números teclee simplemente el primer número, un signo más (+) y el segundo número. Puede incluir tranquilamente espacios antes y después del signo más para que el input sea más fácil de leer.

```
>> 3+5
ans =
8
```

Podemos realizar el cálculo de una potencia directamente.

```
>> 100^50
ans =
1.0000e+100
```

A diferencia de una calculadora, cuando trabaja con enteros, MATLAB muestra el resultado exacto incluso cuando tiene más dígitos de los que cabrían a lo ancho de la pantalla. MATLAB devuelve el valor exacto de 100^{50} si se utiliza la función vpa.



```
>> vpa '99^50'  
  
ans =  
  
6.0500606713753665044791996801256e99
```

Al combinar varias operaciones en una misma instrucción han de tenerse en cuenta los criterios de prioridad habituales entre ellas, que determinan el orden de evaluación de la expresión. Véase el siguiente ejemplo:

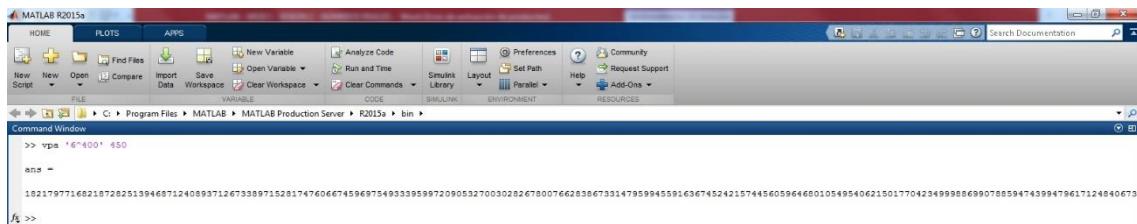
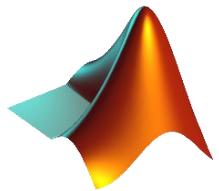
```
>> 2*3^4+(5-1)*6  
  
ans =  
  
186
```

Teniendo en cuenta la prioridad de los operadores, el primero en ser evaluado es el de potenciación. El orden de evaluación normal puede alterarse agrupando expresiones entre paréntesis.

Números enteros

En MATLAB todas las operaciones usuales con números enteros se realizan de forma exacta, independientemente del tamaño que tenga la salida del resultado. Si queremos que el resultado de una operación aparezca en pantalla con un determinado número de cifras exactas, utilizamos el comando de cálculo simbólico `vpa` (variable de precisión aritmética), veamos.

Por ejemplo, al calcular 6 elevado a la potencia 400 con 450 cifras exactas se obtiene como sigue:



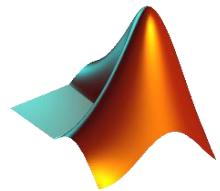
En la imagen la cantidad de cifras no es cubierta aun en pantalla completa, para ello se debe mover la barra de desplazamiento horizontal. La solución exacta para nuestro ejemplo es:

```
>> vpa '6^400' 450
```

```
ans =
```

```
182179771682187282513946871240893712673389715281747606745969754933395997209053270030282678007662838673314795994559163674524215744560596468010549540621501770423499985690788594743994796171248406731
```

El resultado de la operación es exacto, siempre que aparezca un punto al final del resultado. En este caso no son necesarias 450 cifras para expresar el resultado de la operación propuesta, con lo cual la solución es exacta. Si se requiere un número más pequeño de cifras exactas que las que realmente tiene el resultado, MATLAB redondea por la cifra pedida y completa el resultado con potencias de 10. Por ejemplo, para el cálculo anterior con 50 cifras exactas, tenemos.



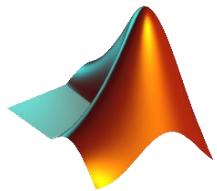
```
>> vpa '6^400' 50
ans =
1.8217977168218728251394687124089371267338971528175e311
```

Funciones con números enteros y divisibilidad

Existen varias funciones en MATLAB con argumento entero, la mayoría de las cuales son relativas a divisibilidad. Entre las cuales se destacan las siguientes:

Función	Significado
rem(n,m)	Resto de la división de n entre m
sign(n)	Signo del n (1 si $n > 0$, 0 si $n = 0$, -1 si $n < 0$)
max(n1,n2)	Máximo de los números n1 y n2
min(n1,n2)	Mínimo de los números n1 y n2.
gcd(n1,n2)	Máximo común divisor de n1 y n2
lcm(n1,n2)	Mínimo común múltiplo de los números n1 y n2
factorial(n)	Factorial de n. $n! = 1 \times 2 \times \dots \times n$
factor(n)	Descompone n en factores primos.

A continuación se presentan algunos ejemplos.



```
>> rem(26,3)           → Resto de la división de 26 entre 3
ans =
2

>> rem(4.1,1.2)        → Resto de la división de 4.1 entre 1.2
ans =
0.5000

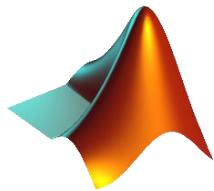
>> rem(-4.1,1.2)
ans =                  → Resto de la división de -4.1 entre 1.2
-0.5000
```

```
>> gcd(1000,gcd(500,625))
ans =                  → Máximo común divisor de
125                   1000, 500 y 625.

>> lcm(1000,lcm(500,625))
ans =                  → Mínimo común múltiplo de
5000                  100, 500 y 625.
```

Operaciones en \mathbb{Q}

Como bien sabemos, los números reales son la unión disjunta de los números racionales y los números irracionales. Un número racional es de la forma p/q , donde p es un entero y q



otro entero. Luego los racionales son números que se pueden representar como el cociente de un entero dividido por otro entero. La forma en que MATLAB trata los racionales es distinta a la de la mayoría de calculadoras. Si se pide a una calculadora que calcule la suma $1/2+1/3+1/4$, la mayoría devolverá algo como 1.0833, que no es más que una aproximación al resultado.

Los números racionales son cocientes de enteros, y MATLAB también puede trabajar con ellos en modo exacto, de manera que el resultado de expresiones en las que intervienen números racionales es siempre otro número racional o entero. Para ello es necesario activar el formato racional con el comando `format rat`. Pero MATLAB también devuelve aproximaciones decimales de los resultados si el usuario así lo desea, activando cualquier otro tipo de formato (por ejemplo, `format short` o `format long`).

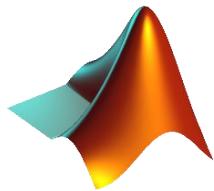
La operación propuesta anteriormente a la calculadora la resuelve MATLAB en modo exacto de la siguiente forma:

```
>> format rat
>> 1/2 + 1/3 + 1/4

ans =
```

13/12

A diferencia de las calculadoras, al hacer operaciones con números racionales el resultado siempre se puede conseguir exacto. Por ello, mientras MATLAB esté tratando con racionales como cocientes de enteros, los mantiene en esta forma. De esta manera, no se arrastran errores de redondeo en los cálculos con fracciones, que pueden llegar a ser muy graves, como demuestra la Teoría de errores. Nótese que, una vez habilitado el formato racional, cuando se pide a MATLAB que sume dos racionales, devuelve un racional como cociente de enteros y así lo representa simbólicamente. Una



vez habilitado el formato de trabajo racional, las operaciones con racionales serán exactas hasta que no se habilite otro formato distinto, en cuyo caso también podemos obtener aproximaciones decimales a los números racionales.

Un número con coma flotante, o sea, un número con punto decimal, se interpreta como exacto siempre que esté habilitado el formato racional. Si hay un número con una coma flotante en la expresión, MATLAB trata toda la expresión como racional exacta y representa el resultado en números racionales. A su vez, si existe un número irracional en una expresión racional, MATLAB lo hace corresponder a una fracción para trabajar en formato racional.

```
>> format rat
>> sqrt(2)/3 + 5/9

ans =

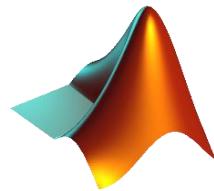
419/408
```

El otro subconjunto fundamental dentro de los números reales es el de los números irracionales, que por su especial naturaleza siempre han generado dificultades en los procesos de cálculo numérico. La imposibilidad de representar un irracional de forma exacta en modo numérico (usando las diez cifras del sistema de numeración decimal) es la causa de la mayoría de los problemas. MATLAB representa los resultados con la mayor precisión que puede o con la precisión requerida por el usuario. Los irracionales no se pueden representar exactamente como la razón entre dos enteros. Si queremos hallar $\sqrt{5}$, MATLAB devuelve la cantidad 2.2361 para el formato por defecto.

```
>> sqrt(5)

ans =

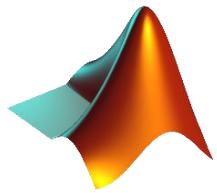
2.2361
```



Existe un grupo importante de números irracionales y reales en general que por su utilización muy común merecen trato especial. MATLAB incorpora los siguientes:

Función	Significado
pi	Número $\pi = 3.1415926$
exp(1)	Número $e = 2.7182818$
inf	Infinito (por ejemplo 1/0)
NaN	Indeterminación (por ejemplo 0/0)
realmin	Menor número real positivo utilizable
realmax	Mayor número real positivo utilizable

A continuación se ilustran estos números con salidas de MATLAB.



MATLAB R2015a

HOME PLOTS APPS

C:\> Users > investigacion > Documents > MATLAB

Current F... Name

- ejemplo...
- ejemplo...
- funcion...
- funcion...
- funcion...
- funcion...
- funcion...
- funcion...
- program...
- program...
- program...
- program...
- program...

Command Window

```
>> format long
>> pi
ans =
    3.141592653589793
>> exp(1)
ans =
    2.718281828459046
>> 1/0
ans =
    Inf
>> 0/0
ans =
    NaN
>> realmin
ans =
    2.225073858507201e-308
>> realmax
ans =
    1.797693134862316e+308
fx >> |
```

>> format long
>> pi
ans =
3.141592653589793

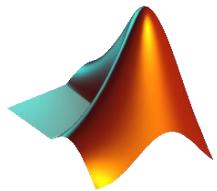
>> 1/0
ans =
Inf

>> 0/0
ans =
NaN

>> realmin
ans =
2.225073858507201e-308

>> realmax
ans =
1.797693134862316e+308





Funciones con argumento real

El conjunto de los números reales es la unión disjunta del conjunto de los números racionales y del conjunto de los números irracionales. Como a su vez el conjunto de los números racionales contiene al conjunto de los números enteros, todas las funciones aplicables a números reales serán válidas también para números irracionales, racionales y enteros. MATLAB dispone de una gama muy completa de funciones predefinidas, la mayoría de las cuales se estudian en capítulos posteriores de este libro. Dentro del grupo de funciones con argumento real que ofrece MATLAB, las más importantes son las siguientes:

- **Funciones trigonométricas**

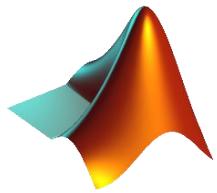
Función	Inversa
$\sin(x)$	$\text{asin}(x)$
$\cos(x)$	$\text{acos}(x)$
$\tan(x)$	$\text{atan}(x)$
$\cot(x)$	$\text{acot}(x)$
$\sec(x)$	$\text{asec}(x)$
$\csc(x)$	$\text{acsc}(x)$

```
>> cos(pi/4)
ans =
    0.7071

>> 2*sin(pi/4)*cos(pi/4)
ans =
    1

>> tan(10)
ans =
    0.6484

>> asin(1)
ans =
    1.5708
```



- **Funciones hiperbólicas**

Función	Inversa
$\sinh(x)$	$\text{asinh}(x)$
$\cosh(x)$	$\text{acosh}(x)$
$\tanh(x)$	$\text{atanh}(x)$
$\coth(x)$	$\text{acoth}(x)$
$\text{sech}(x)$	$\text{asech}(x)$
$\text{csch}(x)$	$\text{acsch}(x)$

```
>> sinh(1)
```

```
ans =
```

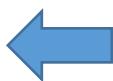
```
1.1752
```

```
>> cosh(pi)
```

```
ans =
```

```
11.5920
```

Recordemos:



$$\sinh(x) = \frac{e^x - e^{-x}}{2}$$



$$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Recordemos:

$$\text{sech}(x) = \frac{1}{\cosh(x)} \quad \rightarrow$$

$$\text{csch}(x) = \frac{1}{\sinh(x)} \quad \rightarrow$$

```
>> asech(0.8)
```

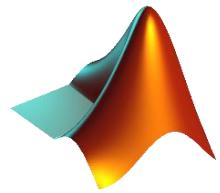
```
ans =
```

```
0.6931
```

```
>> acsch(5)
```

```
ans =
```

```
0.1987
```



- **Funciones exponenciales y logarítmicas**

Función	Significado
exp(x)	Función exponencial en base e (e^x)
log(x)	Función logaritmo en base e de x.
log10(x)	Función logaritmo en base 10 de x.
log2(x)	Función logaritmo en base 2 de x.
sqrt(x)	Función raíz cuadrada de x.

```
>> exp(2)
```

```
ans =
```

```
7.3891
```

```
>> sqrt(25)
```

```
ans =
```

```
5
```

```
>> log(10)
```

```
ans =
```

```
2.3026
```

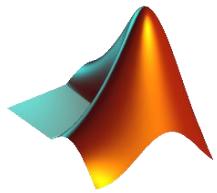
```
>> log10(100)
```

```
ans =
```

```
2
```

- **Funciones específicas de una variable numérica**

Función	Significado
abs(x)	Valor absoluto de real x.
floor(x)	El mayor entero menor o igual que x.
ceil(x)	El menor entero mayor o igual que x.
round(x)	El entero más próximo al real x
fix(x)	Elimina la parte decimal de x
rem(a,b)	Resto de la división entera a y b
sign(x)	Signo del número real x (1 si $x > 0$, 0 si $x = 0$, -1 si $x < 0$)



```
>> sign(5)
```

```
ans =
```

```
1
```

```
>> sign(0)
```

```
ans =
```

```
0
```

```
>> sign(-9)
```

```
ans =
```

```
-1
```

$$\text{sign}(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x = 0 \\ -1 & \text{si } x < 0 \end{cases}$$

```
>> floor(3.2)
```

```
ans =
```

```
3
```

```
>> floor(3.6)
```

```
ans =
```

```
3
```

```
>> ceil(6.1)
```

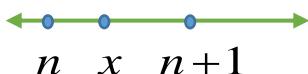
```
ans =
```

```
7
```

```
>> ceil(6.7)
```

```
ans =
```

```
7
```



```
>> round(2.2)
```

```
ans =
```

```
2
```

```
>> round(2.8)
```

```
ans =
```

```
3
```

```
>> fix(1.4)
```

```
ans =
```

```
1
```

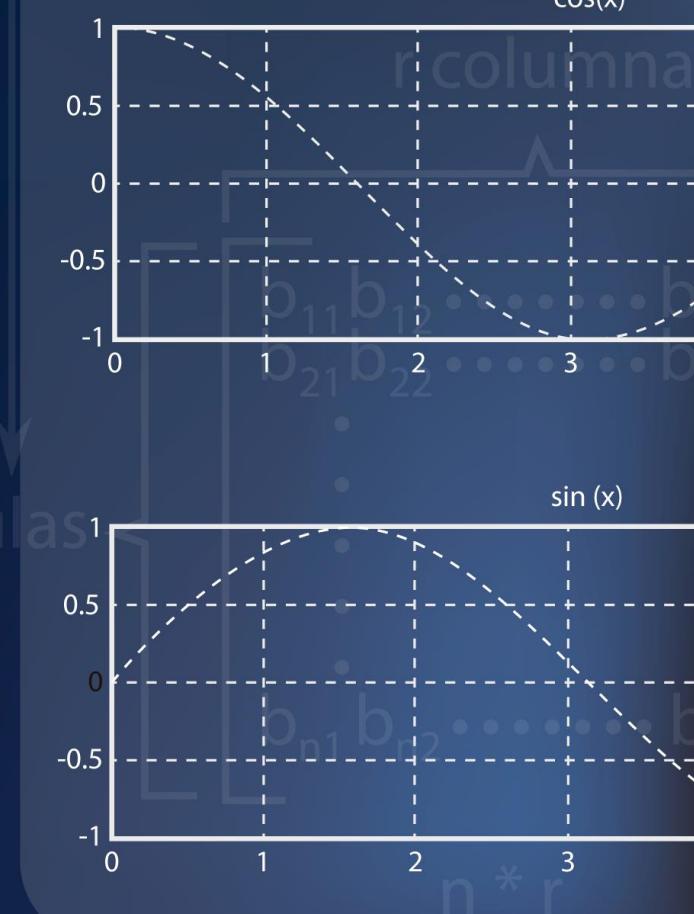
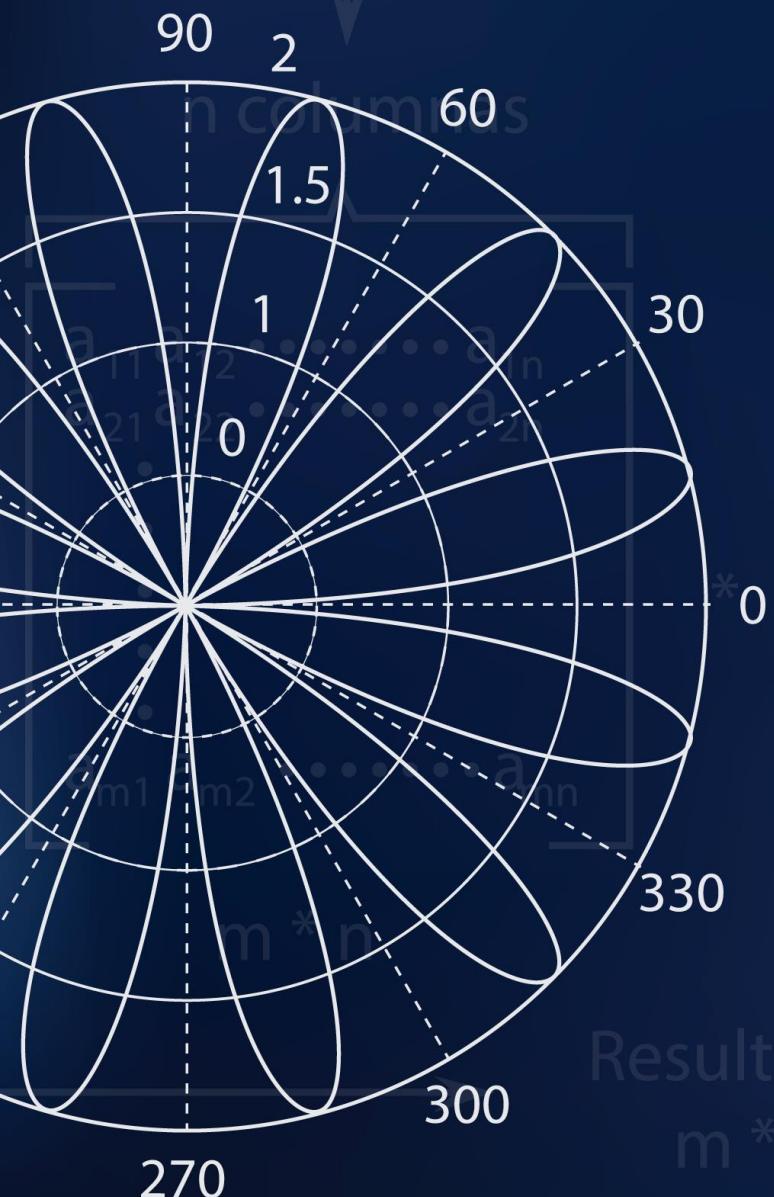
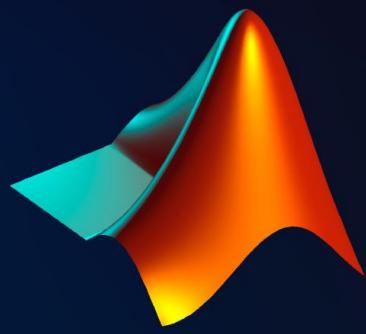
```
>> fix(6.5)
```

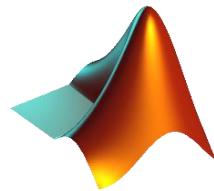
```
ans =
```

```
6
```

Curso MATLAB

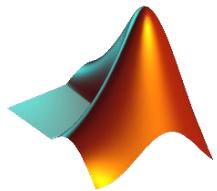
Básico





Contenido

Operaciones en \square	3
Funciones con argumento complejo	4
• Funciones trigonométricas	4
• Funciones hiperbólicas.....	5
• Funciones exponenciales y logarítmicas	5
• Funciones específicas para la parte real e imaginaria	6
Funciones elementales que admiten como argumento un vector complejo v	9
Funciones elementales que admiten como argumento una matriz compleja Z	10
• Trigonométricas.....	10
• Exponenciales.....	11
• Complejas	11
• Numéricas.....	11



Sesión 3: Números Complejos

Operaciones en \mathbb{C}

El trabajo en el campo de los números complejos está perfectamente implementado en MATLAB. Siguiendo la convención de que todas las funciones empiezan con minúscula, una i o una j minúsculas representan el número imaginario $\sqrt{-1}$, que es el valor clave en todo el análisis de variable compleja. Sobre los números complejos pueden ser aplicados los operadores habituales, además de algunas funciones específicas. Tanto la parte real como la parte imaginaria de los números complejos pueden ser constantes, simbólicas o cualquier número real, y las operaciones con ellos se realizan siempre en modo exacto, a no ser que en alguna intervenga una aproximación decimal, en cuyo caso se devuelve una aproximación del resultado.

Como la unidad imaginaria se representa mediante los símbolos i o j, los números complejos se expresan en la forma $a+bi$ o $a+bj$. Es notorio el hecho de no necesitar el símbolo del producto (el asterisco) antes de la unidad imaginaria:

```
>> format short
>> i^2

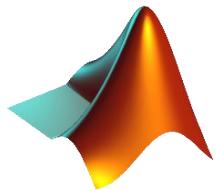
ans =
-1

>> (4-i)*(5+i)/(-1+3i)

ans =
-2.4000 - 6.2000i
```

```
>> format rat
>> (4-i)*(5+i)/(-1+3i)

ans =
-12/5      - 31/5i
```



Funciones con argumento complejo

El trabajo con variable compleja es muy importante en análisis matemático y en sus aplicaciones en ramas importantes de la ingeniería. MATLAB no solamente implementa la posibilidad de operar con números complejos, sino que además incorpora varias funciones con variable compleja. A continuación se presenta un resumen de las más importantes.

- **Funciones trigonométricas**

Función	Inversa
$\sin(z)$	$\text{asin}(z)$
$\cos(z)$	$\text{acos}(z)$
$\tan(z)$	$\text{atan}(z)$
$\cot(z)$	$\text{acot}(z)$
$\sec(z)$	$\text{asec}(z)$
$\csc(z)$	$\text{acsc}(z)$

Command Window

```
>> sin(2i)

ans =

    0.0000 + 3.6269i

>> tan(1+2i)

ans =

    0.0338 + 1.0148i
```

```
>> acos(i/5)

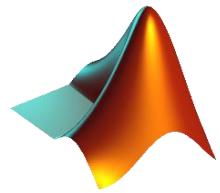
ans =

    1.5708 - 0.1987i

>> acsc(2i/9)

ans =

    0.0000 - 2.2093i
```



- **Funciones hiperbólicas**

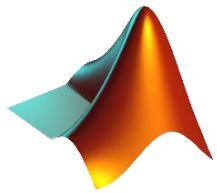
Función	Inversa
$\sinh(z)$	$\text{asinh}(z)$
$\cosh(z)$	$\text{acosh}(z)$
$\tanh(z)$	$\text{atanh}(z)$
$\coth(z)$	$\text{acoth}(z)$
$\text{sech}(z)$	$\text{asech}(z)$
$\text{csch}(z)$	$\text{acsch}(z)$

```
>> cosh(i)  
  
ans =  
  
0.5403  
  
>> sech(0.5+i/4)  
  
ans =  
  
0.9027 - 0.1065i
```

```
>> asinh(3+2i)  
  
ans =  
  
1.9834 + 0.5707i  
  
>> acoth(0.5+2i)  
  
ans =  
  
0.0964 - 0.4442i
```

- **Funciones exponenciales y logarítmicas**

Función	Significado
exp(z)	Función exponencial en base e (e^z)
log(z)	Función logaritmo en base e de z.
log10(z)	Función logaritmo en base 10 de z.
log2(z)	Función logaritmo en base 2 de z.
sqrt(z)	Función raíz cuadrada de z.



```
>> exp(1+i)  
  
ans =  
  
    1.4687 + 2.2874i  
  
>> log(1+2i)  
  
ans =  
  
    0.8047 + 1.1071i  
  
>> sqrt(0.25+3i)  
  
ans =  
  
    1.2768 + 1.1748i
```

Recordar que **log** en Matlab representa al logaritmo neperiano.

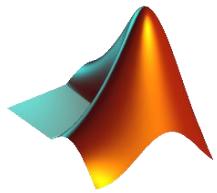
Más aún en □ :

$$\ln(z) = \ln |z| + i\operatorname{Arg}(z)$$

- **Funciones específicas para la parte real e imaginaria**

Función	Significado
floor(z)	Aplica la función floor a real(z) y a imag(z).
ceil(z)	Aplica la función ceil a real(z) y a imag(z).
round(z)	Aplica la función round a real(z) y a imag(z).
fix(z)	Aplica la función fix a real(z) y a imag(z).
abs(z)	Modulo del complejo z.
angle(z)	Argumento del complejo z.
conj(z)	Conjugado del complejo z.
real(z)	Parte real del complejo z.
imag(z)	Parte imaginaria del complejo z.

A continuación se presentan algunos ejemplos de operaciones con números complejos.



```
>> floor(2.7-4.3i)  
ans =  
2.0000 - 5.0000i  
  
>> ceil(5.2-4.1i)  
ans =  
6.0000 - 4.0000i
```

Mayor entero menor o igual en la parte real e imaginaria.

Menor entero mayor o igual en la parte real e imaginaria.

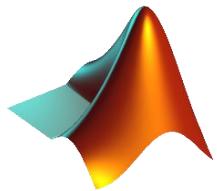
Entero más próximo en la parte real e imaginaria.

Elimina la parte decimal en la parte real e imaginaria.

```
>> round(3.2+3.7i)  
ans =  
3.0000 + 4.0000i  
  
>> fix(-1.6+2.3i)  
ans =  
-1.0000 + 2.0000i
```

```
>> real(3+4i)  
ans =  
3  
  
>> imag(3+4i)  
ans =  
4
```

Muestran la parte real e imaginaria del complejo z.



Muestra el modulo
del complejo z.

```
>> abs(3+4i)
```

```
ans =
```

5

Muestra el argumento
del complejo z.

```
>> angle(3+4i)
```

```
ans =
```

0.9273

Muestra el conjugado
del complejo z.

```
>> conj(3+4i)
```

```
ans =
```

3.0000 - 4.0000i

```
>> real(i^i)
```

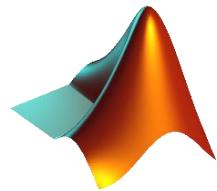
```
ans =
```

0.2079

```
>> (2+2i)^2/(-3+3*sqrt(3)*i)^4
```

```
ans =
```

0.0053 - 0.0031i



Funciones elementales que admiten como argumento un vector complejo v

MATLAB es un software que maneja perfectamente el cálculo vectorial y matricial.

NOTA:

Con el fin de mantener los comandos relacionados a variables complejas juntos, daremos a continuación la lista de los comandos que Matlab nos brinda para vectores y matrices complejos. Los cuales se desarrollaran en las siguientes sesiones con sus temas correspondientes.

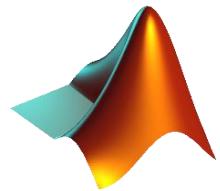
Función	Significado
max(V)	Mayor componente (para complejos se calcula $\max(\text{abs}(V))$)
min(V)	Menor componente (para complejos se calcula $\min(\text{abs}(V))$)
mean(V)	Media de las componentes de V.
median(V)	Mediana de las componentes de V.
std(V)	Desviación típica de las componentes de V.
sort(V)	Ordena de forma ascendente las componentes de V. Para complejos hace la ordenación según los valores absolutos.
sum(V)	Suma las componentes de V.
prod(V)	Multiplica los elementos de V, con lo que $n! = \prod(1:n)$
cumsum(V)	Da el vector de sumas acumuladas de V.
cumprod(V)	Da el vector de productos acumulados de V.

Estas funciones también admiten como argumento una matriz compleja, en cuyo caso el resultado es un vector fila cuyas componentes son los resultados de aplicar la función a cada columna de la matriz.

Funciones elementales que admiten como argumento una matriz compleja Z

- Trigonometrías

Función	Significado
sin(z)	Función seno
sinh(Z)	Función seno hiperbólico
asin(Z)	Función arcoseno
asinh(Z)	Función arcoseno hiperbólico
cos(Z)	Función coseno
cosh(Z)	Función coseno hiperbólico
acos(Z)	Función arcocoseno
acosh(Z)	Función arcocoseno hiperbólico
tan(Z)	Función tangente
tanh(Z)	Función tangente hiperbólica
atan(Z)	Función arcotangente
atanh(Z)	Función arcotangente hiperbólica
sec(Z)	Función secante
sech(Z)	Función secante hiperbólica
asec(Z)	Función arcosecante
asech(Z)	Función arcosecante hiperbólica
csc(Z)	Función cosecante
csch(Z)	Función cosecante hiperbólica
acsc(Z)	Función arcocosecante
acsch(Z)	Función arcocosecante hiperbólica
cot(Z)	Función cotangente
coth(Z)	Función cotangente hiperbólica
acot(Z)	Función arcocotangente
acoth(Z)	Función arcocotangente hiperbólica



- **Exponenciales**

Función	Significado
exp(z)	Función exponencial de base e
log(Z)	Función logaritmo neperiano
log10(Z)	Función logaritmo decimal
sqrt(Z)	Función raíz cuadrada

- **Complejas**

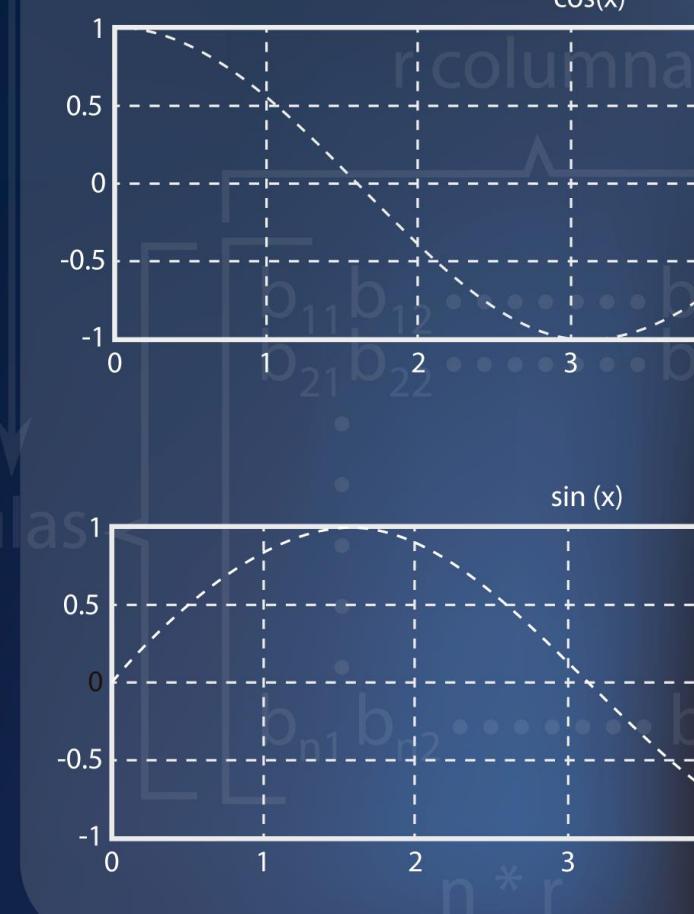
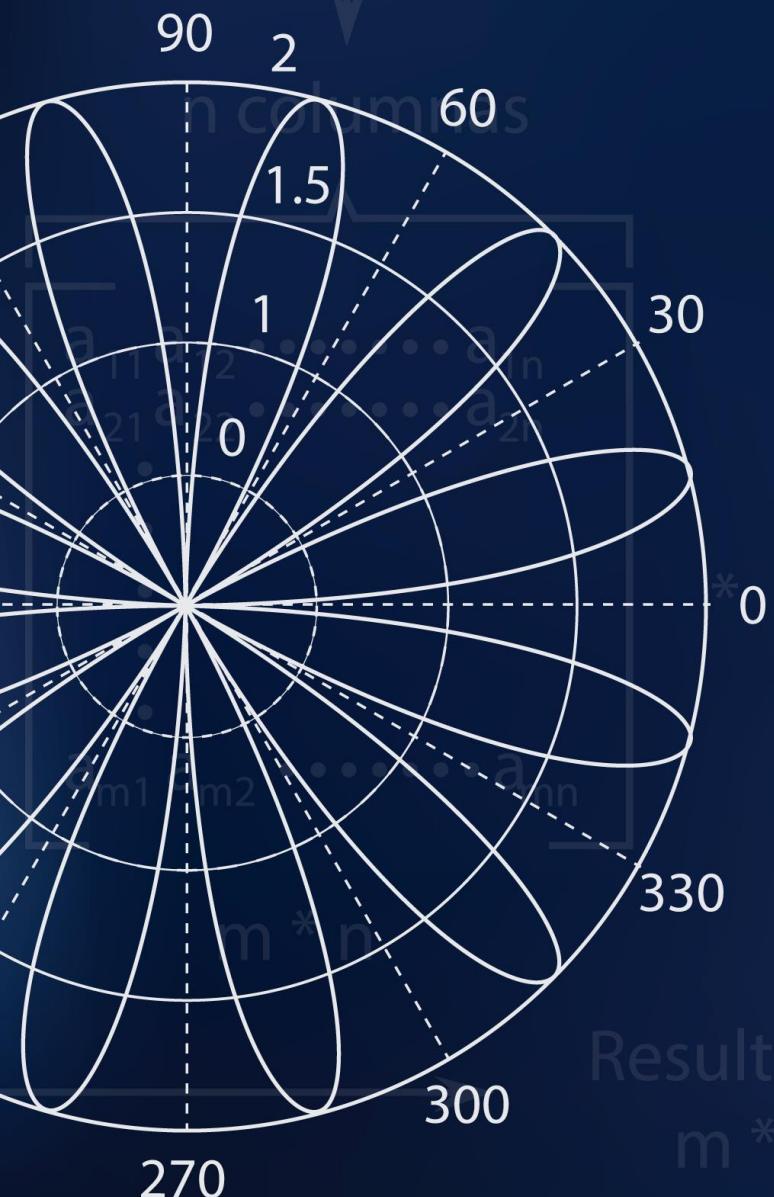
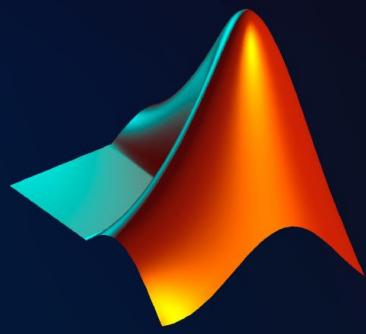
Función	Significado
abs(z)	Modulo o valor absoluto
angle(Z)	Argumento
conj(Z)	Complejo conjugado
imag(Z)	Parte imaginaria
real(Z)	Parte real

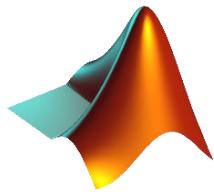
- **Numéricas**

Función	Significado
fix(z)	Elimina las partes decimales
floor(Z)	Redondea los decimales al menor entero más cercano
ceil(Z)	Redondea los decimales al mayor entero más cercano
round(Z)	Efectúa el redondeo común de decimales

Curso MATLAB

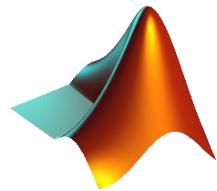
Básico





Contenido

Creación de vectores.....	3
Operaciones con vectores.....	8
Operaciones a un vector	8
Operación elemento a elemento.....	9
Funciones elementales que admiten como argumento un vector complejo v	13



Sesión 4: Vectores

Creación de vectores

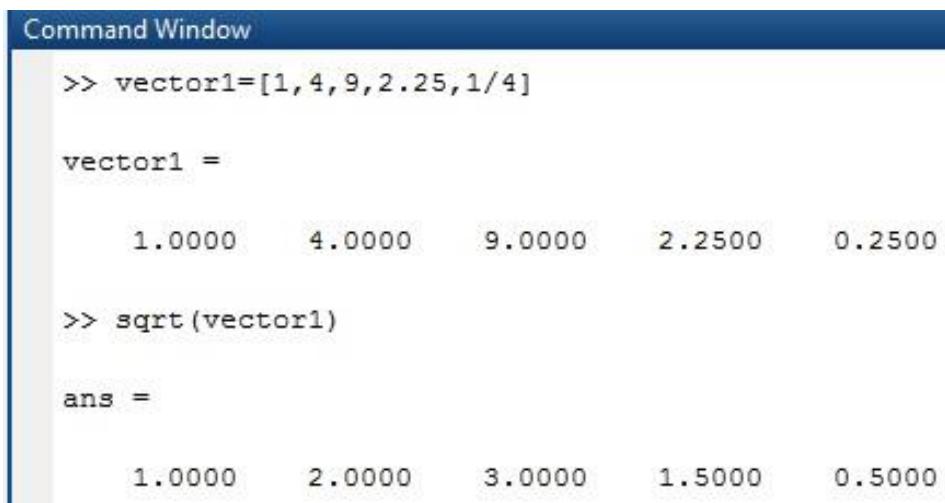
Para representar a un vector de n elementos se puede definir en MATLAB una variable de las siguientes formas:

$v = [v_1, v_2, \dots, v_n]$ es un vector fila de valores v_1, v_2, \dots, v_n .

$v = [v_1 \ v_2 \ \dots \ v_n]$ es un vector fila de valores v_1, v_2, \dots, v_n .

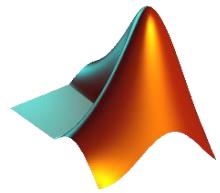
$v = [v_1; v_2; \dots; v_n]$ es un vector columna de valores v_1, v_2, \dots, v_n .

Cuando se aplican la mayoría de los comandos y funciones de MATLAB sobre una variable vectorial el resultado que se obtiene es la aplicación del comando o función sobre cada elemento del vector:



```
Command Window
>> vector1=[1,4,9,2.25,1/4]
vector1 =
    1.0000    4.0000    9.0000    2.2500    0.2500
>> sqrt(vector1)
ans =
    1.0000    2.0000    3.0000    1.5000    0.5000
```

Existen diferentes formas de definir una variable vectorial sin necesidad de explicitar entre corchetes todos sus elementos separados por comas o espacios en blanco.



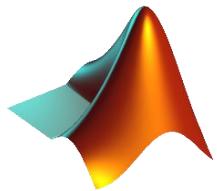
variable=[a:b] variable=(a:b) variable=a:b	Define el vector que comienza en a y los que le siguen en posición se van incrementando en una unidad sin llegar a exceder el valor de b .
variable=[a:s:b] variable=(a:s:b) variable=a:s:b	Define el vector que comienza en a y los que le siguen en posición se van incrementando en una cantidad s , sin llegar a exceder el valor de b .
variable=linspace(a,b,n)	Define un vector con n elementos uniformemente espaciados entre si que van desde a hasta b .
variable=logspace(a,b,n)	Define un vector con n elementos uniformemente espaciados en escala logarítmica.

Veamos, algunos ejemplos:

```
>> vector2=[5:5:25]
vector2 =
      5     10     15     20     25
```

Hemos definido un vector con los números del 5 al 25 con incremento de 5 unidades.

Generemos un vector con elementos entre 10 y 30 separados una unidad.



```
>> vector3=[10:30]

vector3 =

Columns 1 through 12

10    11    12    13    14    15    16    17    18    19    20    21

Columns 13 through 21

22    23    24    25    26    27    28    29    30
```

Creemos un vector con 6 elementos igualmente espaciados que van desde 0 a 20.

```
>> vector4=linspace(10,30,6)

vector4 =

10    14    18    22    26    30
```

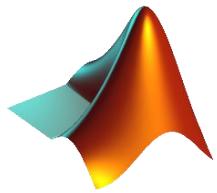
A continuación se muestra un vector con 5 elementos logarítmicamente espaciados. Estos van desde 10^a hasta 10^b , donde $a=1$ y $b=2$.

```
>> vector5=logspace(1,2,5)

vector5 =

10.0000    17.7828    31.6228    56.2341   100.0000
```

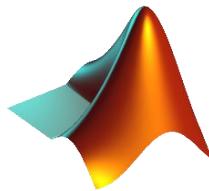
Como vimos al inicio, en MATLAB también existe la posibilidad de considerar vectores fila y vectores columna. Un vector columna se obtiene separando sus elementos por punto y coma, o también transponiendo un vector fila mediante una comilla simple situada al final de su definición.



```
>> a=[10;20;30;40]  
  
a =  
  
    10  
    20  
    30  
    40  
  
>> a=(10:14); b=a'  
  
b =  
  
    10|  
    11  
    12  
    13  
    14  
  
>> c=(a')'  
  
c =  
  
    10    11    12    13    14
```

Asimismo podemos seleccionar un elemento de un vector o un subconjunto de elementos. Conozcamos su sintaxis:

x(n)	Devuelve el n -ésimo elemento del vector x .
x(a:b)	Devuelve los elementos del vector x que se encuentran entre el a -ésimo y el b -ésimo elemento, ambos inclusive.
x(a:p:b)	Devuelve un vector con primer elemento igual a la posición a -ésima, seguido de las posiciones que se incrementan de p en p unidades, sin exceder la posición b -ésima. (a>b).
x(b:-p:a)	Devuelve un vector con primer elemento igual



	a la posición b -ésima, seguido de las posiciones que decrecen de p en p unidades, sin llegar a ser menor de la posición a -ésima (b>a).
x([a b c])	Devuelve los a -ésimo, b -ésimo y c -ésimo elementos del vector x .

Veamos algunos ejemplos:

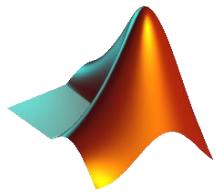
```
>> x=(1:10)  
x =  
1 2 3 4 5 6 7 8 9 10
```

Aquí se creó un vector con elementos del 1 al 10.

```
>> x(6)  
ans =  
6  
  
>> x(4:7)  
ans =  
4 5 6 7
```

Muestra el sexto elemento del vector x.

Muestra los elementos del vector x, desde la posición 4 a la 7.



```
>> x(2:3:9)
```

```
ans =
```

```
2 5 8
```

```
>> x(9:-3:2)
```

```
ans =
```

```
9 6 3
```

Muestra los elementos del vector x, desde la posición 2 a la 9 con incrementos de 3 posiciones.

Muestra los elementos del vector x, desde la posición 9 a la 2 con decrecimientos de 3 posiciones.

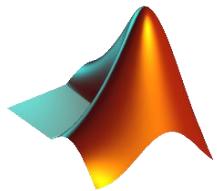
Operaciones con vectores

En Matlab podemos hacer operaciones con los elementos de un vector y también entre vectores como en un espacio vectorial.

- **Operaciones a un vector**

x+n	Suma el número real n a cada elemento del vector x .
x-n	Resta el número real n a cada elemento del vector x .
n*x	Multiplica por n a cada elemento del vector x .
(1/n)*x	Divide por n a cada elemento del vector x .
x.^n	Eleve cada elemento del vector x a la “ n ”, n número real.

Definamos x, y luego veamos algunos ejemplos:



```
>> x=1:5  
  
x =  
1 2 3 4 5  
  
>> x-2  
  
ans =  
-1 0 1 2 3  
  
>> x+3  
  
ans =  
4 5 6 7 8  
  
>> 2*x  
  
ans =  
2 4 6 8 10
```

Define x, de 1 a 5.

Resta 2 a cada elemento de x.

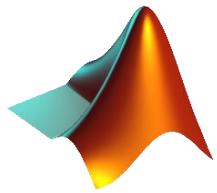
Suma 3 a cada elemento de x.

Multiplica por 3 a cada elemento de x

- **Operación elemento a elemento**

x-y	Resta de vectores	x-y=[x(i)-y(i)]
x+y	Suma de vectores	x+y=[x(i)+y(i)]
x.*y	Producto elemento a elemento	x.*y=[x(i)*y(i)]
x./y	Cociente elemento a elemento	x./y=[x(i)/y(i)]
x.\y	Cociente elemento a elemento	x.\y=[y(i)/x(i)]
x.^y	Potenciación elemento a elemento	x.^y=[x(i)^y(i)]

Ahora definamos “y”, y operemos junto con “x”



```
>> y=5:-1:1
```

```
y =
```

```
5 4 3 2 1
```

Define **y**, de 5 a 1 decreciendo en una unidad.

```
>> x-y
```

```
ans =
```

```
-4 -2 0 2 4
```

Resta elemento a elemento, **x** con **y**.

```
>> x+y
```

```
ans =
```

```
6 6 6 6 6
```

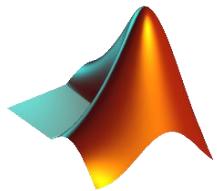
Suma elemento a elemento, **x** con **y**.

```
>> x.*y
```

```
ans =
```

```
5 8 9 8 5
```

Multiplica 1-1 los elementos de **x** con **y**.



```
>> x./y
ans =
0.2000    0.5000    1.0000    2.0000    5.0000
← Divide 1-1
elementos
de x con y.

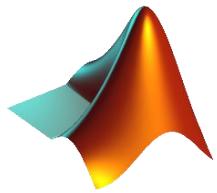
>> x.\y
ans =
5.0000    2.0000    1.0000    0.5000    0.2000
← Divide 1-1
elementos
de y con x.

>> x.^y
ans =
1    16    27    16     5
← Eleva 1-1
elementos
de x con y.
```

Notemos que es importante el uso del punto (.) en las operaciones especificadas con vectores de la misma dimensión, pues sino Matlab nos devolverá error, veamos:

```
>> x*y
Error using *
fx Inner matrix dimensions must agree.
```

Sin embargo, veamos que esta operación es posible si para este ejemplo “y” fuera un vector columna, pues estaría bien definido el producto dado que “x” tiene 5 filas mientras que y tiene 5 columnas, tratándolas como matrices. Tema que se verá en detalle en las próximas sesiones.

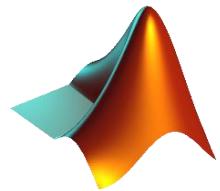


```
>> x=1:5, y=5:-1:1  
  
x =  
  
1 2 3 4 5  
  
y =  
  
5 4 3 2 1  
  
>> x*y'  
  
ans =  
  
35
```

Ahora notemos que el resultado es un número (para nuestro caso 35) y no un vector como quizás podríamos haber esperado. Esto se debe a que en realidad esta operación (*) representa el producto interno (o producto escalar) de los 2 vectores y de hecho solo es posible esta operación sin punto (de las que se especifican en la tabla).

A continuación mostramos 2 puntos muy importantes para el manejo con los resultados de las operaciones hechas con los vectores, pues ellos nos van a permitir analizar y ayudar a la salida del resultado deseado. Dependiendo del problema que estemos abordando también nos permite aislar los elemento que sean o no sean necesarios en las siguientes operaciones a realizar. Y con ello facilitar tanto las operaciones como el tiempo de ejecución de los programas.

a, b	Muestra en pantalla los vectores que tengamos almacenados en la memoria.
c=[a b]	Guarda en el vector , los vectores a y b .

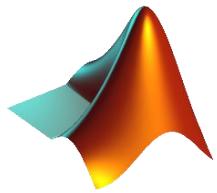


Funciones elementales que admiten como argumento un vector complejo v

MATLAB es un software que maneja perfectamente el cálculo vectorial y matricial.

Su propio nombre, laboratorio matricial, ya da idea de su potencia para el trabajo con vectores y matrices. MATLAB permite trabajar con funciones de variable compleja, pero además esta variable puede ser vectorial e incluso matricial. A continuación se presenta una tabla con las funciones de variable compleja vectorial que incorpora MATLAB.

Función	Significado
max(V)	Mayor componente (para complejos se calcula max(abs(V)))
min(V)	Menor componente (para complejos se calcula min(abs(V)))
mean(V)	Media de las componentes de V .
median(V)	Mediana de las componentes de V .
std(V)	Desviación típica de las componentes de V .
sort(V)	Ordena de forma ascendente las componentes de V . Para complejos hace la ordenación según los valores absolutos.
sum(V)	Suma las componentes de V .
prod(V)	Multiplica los elementos de V , con lo que $n! = prod(1:n)$
cumsum(V)	Da el vector de sumas acumuladas de V .
cumprod(V)	Da el vector de productos acumulados de V .



```
>> v=[5 3 1 10 8 6], w=[2+3i, -3+i, 2i, 4-i, -7-5i, 6]

v =
      5      3      1     10      8      6

w =
Columns 1 through 3

    2.0000 + 3.0000i   -3.0000 + 1.0000i   0.0000 + 2.0000i

Columns 4 through 6

    4.0000 - 1.0000i   -7.0000 - 5.0000i   6.0000 + 0.0000i

>> max(v), max(w)

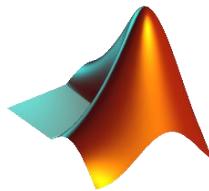
ans =
      10

ans =
      -7.0000 - 5.0000i

>> min(v), min(w)

ans =
      1

ans =
      0.0000 + 2.0000i
```



```
>> mean(v), mean(w)

ans =

5.5000

ans =
0.3333

>> median(v), median(w)

ans =

5.5000
```

```
ans =
3.0000 + 1.0000i
```

```
>> sort(v), sort(w)
```

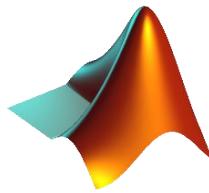
```
ans =
1 3 5 6 8 10
```

```
ans =
Columns 1 through 3
0.0000 + 2.0000i -3.0000 + 1.0000i 2.0000 + 3.0000i
Columns 4 through 6
4.0000 - 1.0000i 6.0000 + 0.0000i -7.0000 - 5.0000i
```

```
>> std(v), std(w)

ans =
3.2711

ans =
5.5377
```



```
>> sum(v), sum(w)

ans =

33

ans =

2

>> prod(v), prod(w)

ans =

7200

ans =

-4.1760e+03 + 2.4720e+03i

>> cumsum(v), cumsum(w)

ans =

5      8      9     19     27     33

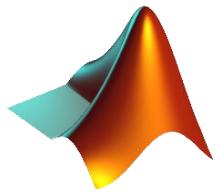
ans =

Columns 1 through 3

2.0000 + 3.0000i  -1.0000 + 4.0000i  -1.0000 + 6.0000i

Columns 4 through 6

3.0000 + 5.0000i  -4.0000 + 0.0000i  2.0000 + 0.0000i
```



```
>> cumprod(v), cumprod(w)

ans =

Columns 1 through 5

      5          15          15         150        1200

Column 6

    7200

ans =

1.0e+03 *

Columns 1 through 3

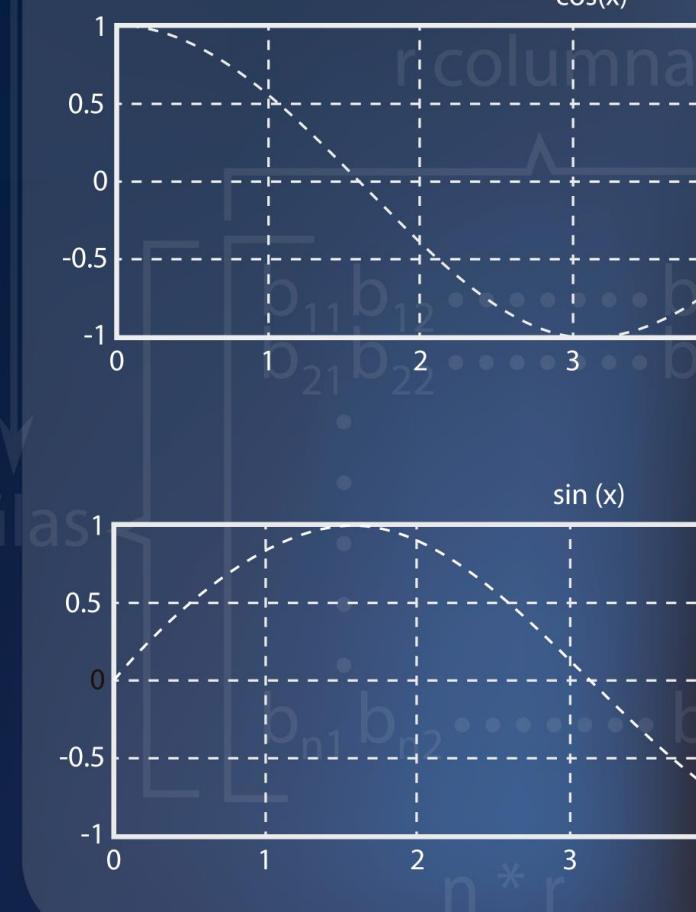
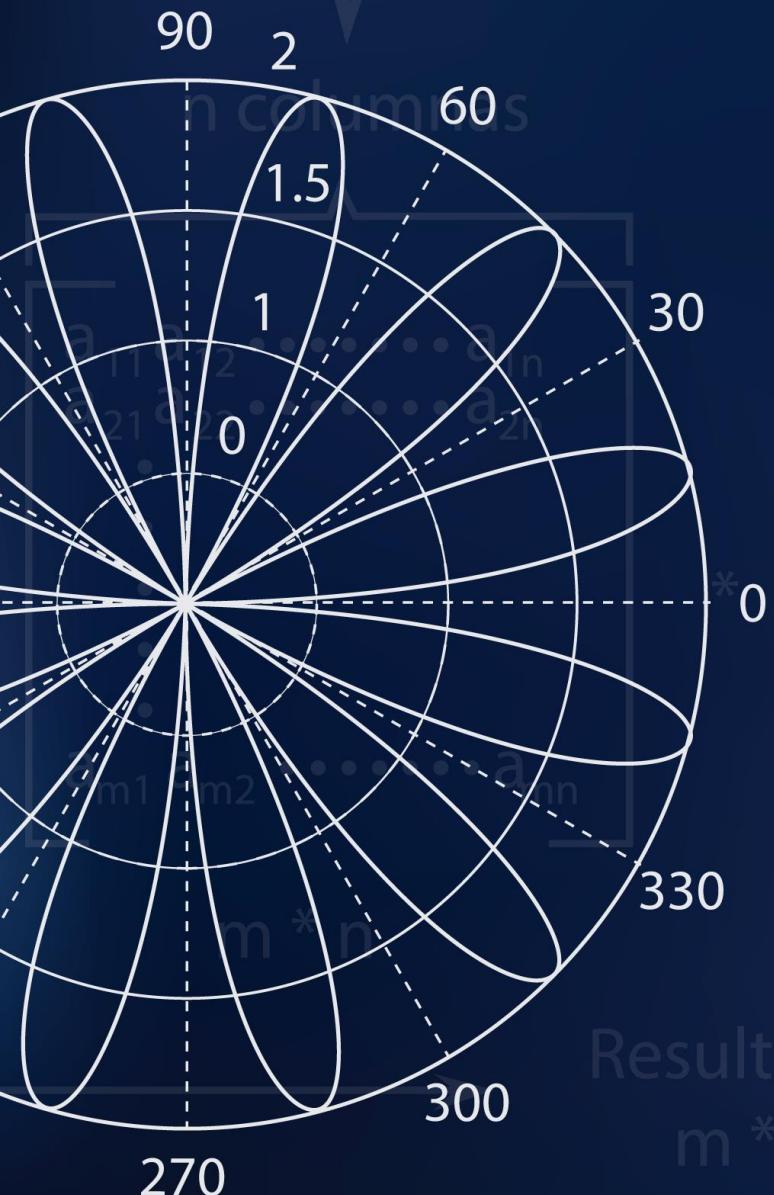
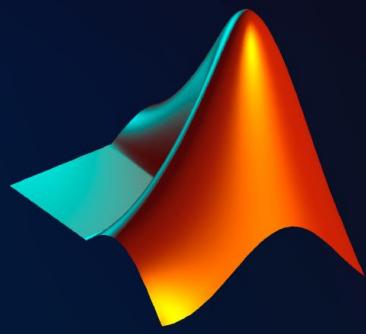
  0.0020 + 0.0030i  -0.0090 - 0.0070i   0.0140 - 0.0180i

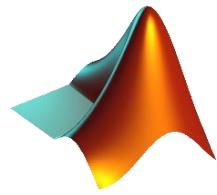
Columns 4 through 6

  0.0380 - 0.0860i  -0.6960 + 0.4120i  -4.1760 + 2.4720i
```

Curso MATLAB

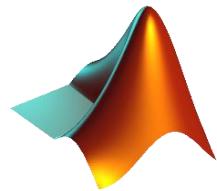
Básico





Contenido

Evaluando en la variable	3
Hallando sus raíces	5
Producto y División polinomial	5
• Producto polinomial.....	5
• División polinomial	7
Derivada e integración de polinomios.....	7
• Derivada de un polinomio	7
• Integración de un polinomio	8
Polinomio interpolador	9



Sesión 5: Polinomios

Evaluando en la variable

Un polinomio $P(x) = a_1 x^n + a_2 x^{n-1} + \dots + a_n x + a_{n+1}$ puede ser introducido en Matlab mediante un vector de longitud **n+1** cuyos elementos son los coeficientes del polinomio completo y ordenado (los monomios se escriben de mayor a menor grado y se encuentran todos los términos completando con cero si fuese necesario).

```
>> P=[a1 a2 ... an+1]
```

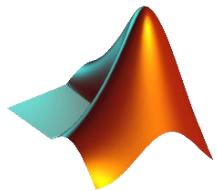
Ejemplo: Definamos el polinomio $P(x) = x^3 + 2x^2 - 3$ en Matlab

Command Window

```
>> p=[1 2 0 -3]
p =
    1     2     0    -3
```

Para evaluar un polinomio haremos uso de **polyval(p,x)**. Hallemos $P(2)$

```
>> y=polyval(p,2)
y =
    13
```



También podemos hacer la evaluación sobre números complejos, por ejemplo encontremos el valor $P(-3+i)$

```
>> y=polyval(p,-3+i)

y =

-5.0000 +14.0000i
```

Asimismo podemos definir, polinomios con coeficiente complejos. Y evaluarlos en los reales o complejos.

Ejemplo: Sea $S(x)=2x^3+ix^2+x-3+i$, hallemos $S(-2+i)$

```
>> s=[2 i 1 -3+i]

s =

Columns 1 through 3

2.0000 + 0.0000i 0.0000 + 1.0000i 1.0000 + 0.0000i

Column 4

-3.0000 + 1.0000i

>> y1=polyval(s,2.5)

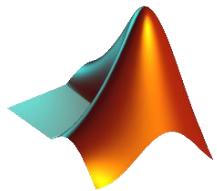
y1 =

30.7500 + 7.2500i

>> y2=polyval(s,-2+i)

y2 =

-5.0000 +27.0000i
```



Hallando sus raíces

Tanto para los polinomios con coeficientes reales $P(x)$ como también para polinomios con coeficientes complejos $S(x)$, Matlab nos permite encontrar sus raíces con el uso del comando **roots()**.

Ejemplo: Hallemos las raíces para los polinomios definidos anteriormente

```
>> raices_p=roots(p)

raices_p =
-1.5000 + 0.8660i
-1.5000 - 0.8660i
1.0000 + 0.0000i

>> raices_s=roots(s)

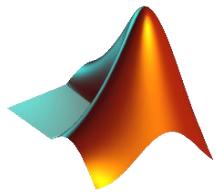
raices_s =
-0.4185 + 1.0364i
-0.5693 - 1.2533i
0.9878 - 0.2832i
```

Producto y División polinomial

- **Producto polinomial**

Ahora definamos el polinomio $T(x)=x^2-x+5$, para multiplicar los polinomios $P(x)$ y $T(x)$ usaremos el comando **conv(p,t)**.

Primero definamos en Matlab $T(x)$



```
>> t=[1 -1 5]
t =
    1     -1      5

>> P1=conv(p,t)
P1 =
    1      1      3      7      3     -15
```

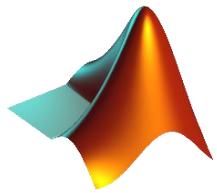
El resultado nos dice que el producto es el siguiente polinomio
 $P_1(x) = x^5 + x^4 + 3x^3 + 7x^2 + 3x - 15$.

De la misma manera podemos obtener el resultado de la multiplicación de un polinomio con coeficientes reales y otro con coeficientes complejos. Como veremos a continuación.

```
>> P2=conv(s,t)
P2 =
    Columns 1 through 3
    2.0000 + 0.0000i   -2.0000 + 1.0000i   11.0000 - 1.0000i
    Columns 4 through 6
   -4.0000 + 6.0000i    8.0000 - 1.0000i  -15.0000 + 5.0000i
```

Aquí el polinomio resultante es el siguiente:

$$P_2(x) = 2x^5 + (-2+i)x^4 + (11-i)x^3 + (-4+6i)x^2 + (8-i)x - 15 + 5i$$



- **División polinomial**

Si queremos dividir dos polinomios $P(x)$ y $T(x)$, Matlab nos brinda el comando **[q,r]=deconv(p,t)** y nos devolverá el cociente $q(x)$ y residuo $r(x)$.

```
>> [q,r]=deconv(p,t)

q =
    1     3

r =
    0     0    -2   -18
```

Esto significa que los polinomios cociente y residuo son:

$$Q(x) = x + 3 \quad \text{y} \quad R(x) = -2x - 18$$

Derivada e integración de polinomios

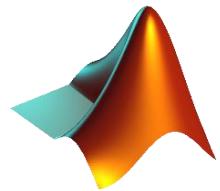
- **Derivada de un polinomio**

Siguiendo con el polinomio $P(x) = x^3 + 2x^2 - 3$, podemos calcular su derivada en base al comando **polyder(p)**

```
>> der=polyder(p)

der =
    3     4     0
```

$$\text{Esto es } \frac{dP}{dx} = 3x^2 + 4x$$



• Integración de un polinomio

Como vimos la derivada de un polinomio fue calculada correctamente, ahora conozcamos como hallar la integral para $P(x)=x^3+2x^2-3$. Necesitaremos para ello emplear el comando **polyint(p)**

```
>> I=polyint (p)

I =

    0.2500    0.6667         0    -3.0000         0
```

Esto es, $\int P(x)dx=0.25x^4+0.6667x^3-3x$. Notemos que Matlab por defecto asume como constante de integración al cero.

Por otro lado, es recomendable usar el formato para racionales y poder así tener una idea más clara.

```
>> format rat
>> I=polyint (p)

I =

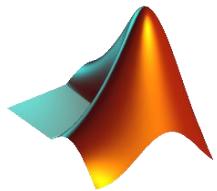
    Columns 1 through 4

    1/4          2/3          0         -3

    Column 5

    0
```

De esta forma, claramente obtenemos el resultado esperado para nuestro ejemplo: $\int P(x)dx=\frac{1}{4}x^4+\frac{2}{3}x^3-3x$.



Polinomio interpolador

Si queremos hacer un ajuste de curva polinómica a partir de unos puntos, podemos usar la función **polyfit(x,y,n)** que nos devuelve los coeficientes de un polinomio $P(x)$ de grado n que tiene un mejor ajuste para los valores en **(x,y)**.

Ejemplo: primero generemos un vector **x** con elementos que van de **[0 4pi]**. Estas serán las abscisas.

```
>> x=linspace(0,4*pi,10)

x =

Columns 1 through 6

    0    1.3963    2.7925    4.1888    5.5851    6.9813

Columns 7 through 10

    8.3776    9.7738   11.1701   12.5664
```

Y ahora generemos las ordenadas, digamos que están definidas de la siguiente forma.

```
>> y=sin(x)

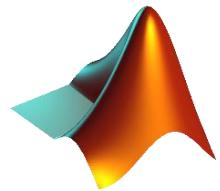
y =

Columns 1 through 6

    0    0.9848    0.3420   -0.8660   -0.6428    0.6428

Columns 7 through 10

    0.8660   -0.3420   -0.9848   -0.0000
```



Ya teniendo los puntos a los cuales queremos interpolar.
Usamos el comando **polyfit**, del siguiente modo.

```
>> p=polyfit(x,y,7)

p =

    Columns 1 through 6

    -0.0001    0.0028   -0.0464    0.3702   -1.3808    1.9084

    Columns 7 through 8

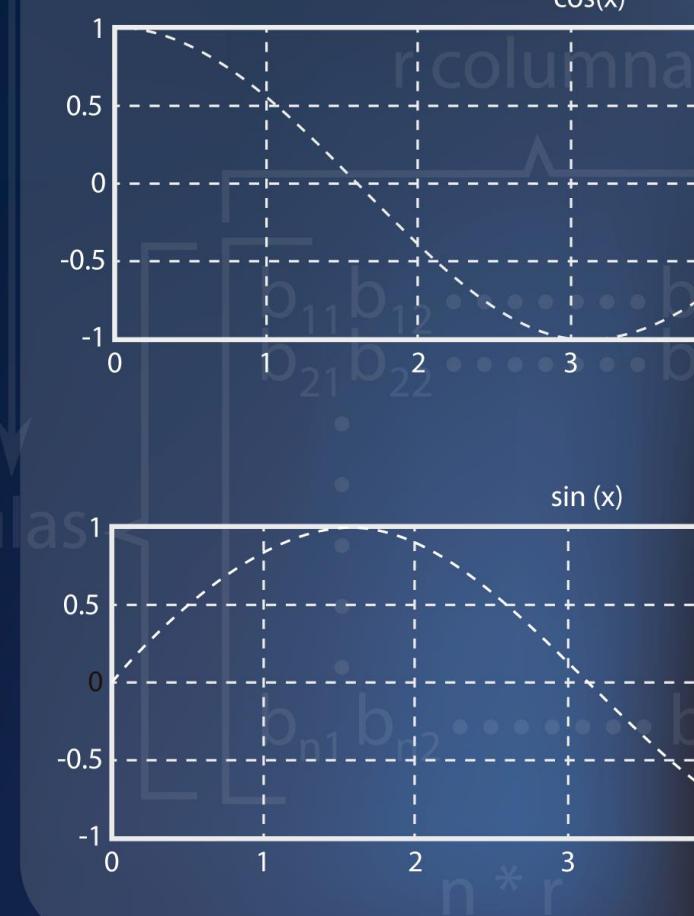
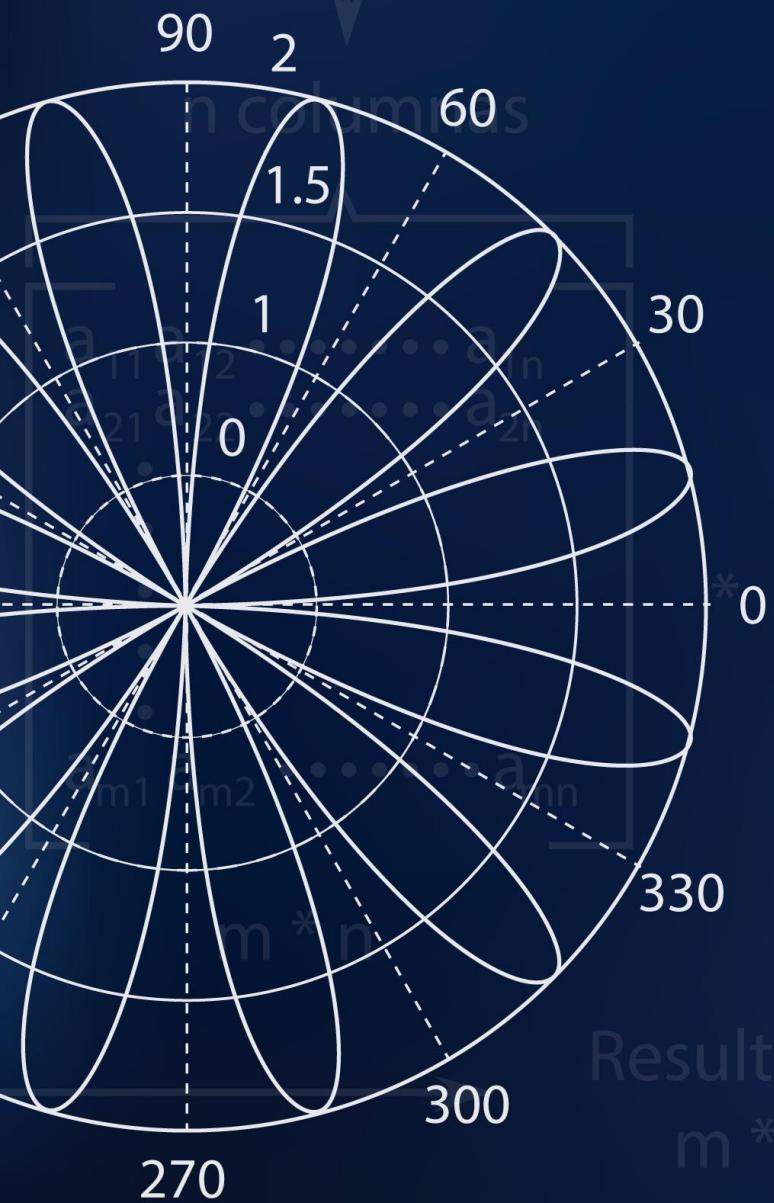
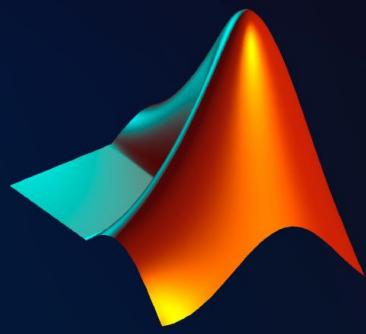
    -0.1141    0.0002
```

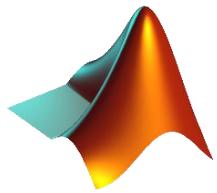
Por lo tanto el polinomio que pasa por esos puntos es:

$$P(x) = -0.0001x^7 + 0.0028x^6 - 0.0464x^5 + 0.3702x^4 - 1.3808x^3 + 1.9084x^2 - 0.1141x + 0.0002$$

Curso MATLAB

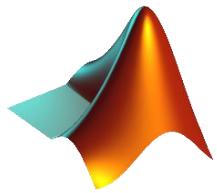
Básico





Contenido

Matrices y Submatrices	3
Operaciones con matrices.....	9
• Operaciones entre matrices.....	9
• Operaciones elemento a elemento.....	11
Características de la matriz	13
Matrices especiales	14
Funciones elementales que admiten como argumento una matriz compleja Z	17
• Trigonométricas.....	17
• Exponenciales.....	18
• Complejas	19
• Numéricas.....	20



Sesión 6: Matrices

Matrices y Submatrices

En matemáticas una matriz la representamos por:

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$$

En Matlab se definen las matrices introduciendo entre corchetes todos sus vectores fila separados por punto y coma. Los vectores se pueden introducir separando sus componentes por espacios en blanco o por comas, tal y como ya sabemos. Por ejemplo, una variable matricial de dimensión 3x3 se puede introducir de las siguientes formas:

$$A = [a_{11} \ a_{12} \ a_{13}; a_{21} \ a_{22} \ a_{23}; a_{31} \ a_{32} \ a_{33}]$$

$$A = [a_{11}, a_{12}, a_{13}; a_{21}, a_{22}, a_{23}; a_{31}, a_{32}, a_{33}]$$

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

```
A =
```

$$\begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{matrix}$$

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

```
A =
```

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix}$$

```
>> A=[1 2 3
```

$$\begin{matrix} 4 & 5 & 6 \end{matrix}$$

```
7 8 9]
```

```
A =
```

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

Una vez que una variable matricial ha sido definida, MATLAB habilita muchos caminos para insertar, extraer, renumerar y manipular en general sus elementos. La tabla siguiente presenta diferentes posibilidades de definición de variables matriciales.

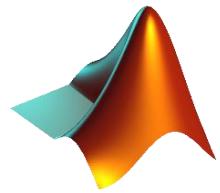
Comando	Significado
A(m,n)	Define el elemento (m,n) de la matriz A.
A(a:b,c:d)	Define la submatriz de A formada por las filas que hay entre la a-ésima y la b-ésima y por las columnas que hay entre la c-ésima y la d-ésima.
A(a:p:b,c:q:d)	Define la submatriz de A formada por las filas que hay entre la a-ésima y la b-ésima tomándolas de p en p, y por las columnas que hay entre la c-ésima y la d-ésima tomándolas de q en q.
A([a b],[c d])	Define la submatriz de A formada por la intersección de las filas a-ésima y b-ésima y las columnas c-ésima y d-ésima.
A([a b c ...],[e f g ...])	Define la submatriz de A formada por la intersección de las filas a, b, c, ... y las columnas e, f, g, ...
A(:,c:d)	Define la submatriz de A formada por todas las filas de A y por las columnas que hay entre la c-ésima y la d-ésima.
A(:,[c d e ...])	Define la submatriz de A formada por todas las filas de A y por las columnas c, d, e, ...
A(a:b,:)	Define la submatriz de A formada por todas las columnas de A y por las filas que hay entre la a-ésima y la b-ésima
A([a b c ...],:)	Define la submatriz de A formada por todas las columnas de A y por las filas a, b, c, ...
A(a,:)	Define la fila a-ésima de la matriz A
A(:,b)	Define la columna b-ésima de la matriz

	A
A(:)	Define un vector columna cuyos elementos son las columnas de A situadas por orden una debajo de otra.
A(:, :)	Equivale a toda la matriz A
[A, B, C, ...]	Define la matriz formada por las submatrices A, B, C, ...
SA = []	Borra la submatriz de la matriz A, SA y devuelve el resto

A continuación, vamos a definir la matriz A; a la cual aplicaremos los comandos estudiados en la tabla. Se dejara como ejercicio los comandos que solo cambien su aplicación de columna a fila.

Recordar que con el fin de mantener un orden también podemos usar los puntos suspensivos.

Cambia el valor del elemento (3,4) por 5.



```
>> A=[1 5 9 13 17 21; ...
2 6 10 14 18 22; ...
3 7 11 15 19 23; ...
4 8 12 16 20 24]

A =

1      5      9     13     17     21
2      6     10     14     18     22
3      7     11     15     19     23
4      8     12     16     20     24

>> A(3,4)

ans =

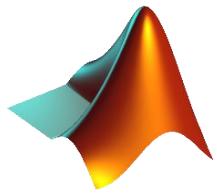
15

>> A(3,4)=5

A =

1      5      9     13     17     21
2      6     10     14     18     22
3      7     11      5     19     23
4      8     12     16     20     24
```

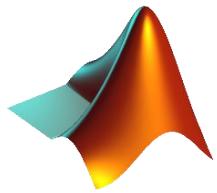
Si solo necesitamos
la columna 4.



```
>> A(1:3,2:6)  
  
ans =  
  
 5     9    13    17    21  
 6    10    14    18    22  
 7    11     5    19    23  
  
>> A(1:2:3,2:2:6)  
  
ans =  
  
 5    13    21  
 7     5    23  
  
>> A([1 3],[2 6])  
  
ans =  
  
 5    21  
 7    23  
  
>> B=A(:,3:5)  
  
B =  
  
 9    13    17  
10    14    18  
11     5    19  
12    16    20  
  
>> C=A(:,[1 5 2])  
  
C =  
  
 1    17     5  
 2    18     6  
 3    19     7  
 4    20     8
```

```
>> A(:,4)  
  
ans =  
  
 13  
 14  
 5  
 16  
  
>> A(:)  
  
ans =  
  
 1  
 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 10  
 11  
 12  
 13  
 14  
 5  
 16  
 17  
 18  
 19  
 20  
 21  
 22  
 23  
 24
```

De matriz a vector



Recordar que fue
modificado el
elemento (3,4)

```
>> A(:,:,1)  
  
ans =  
  
 1   5   9   13   17   21  
 2   6   10  14   18   22  
 3   7   11   5   19   23  
 4   8   12  16   20   24
```

```
>> D=[B,C]
```

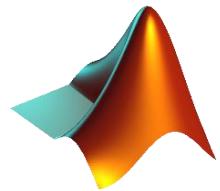
```
D =  
  
 9   13   17   1   17   5  
10   14   18   2   18   6  
11   5   19   3   19   7  
12   16   20   4   20   8
```

```
>> D(:,2:4)=[]
```

```
D =  
  
 9   17   5  
10   18   6  
11   19   7  
12   20   8
```

Las matrices B y C,
se definieron con
anterioridad.

Consideremos
como submatriz la
que se forma con
las columnas 2 a 4.

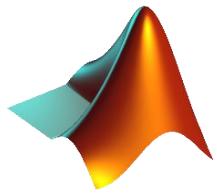


Operaciones con matrices

Las operaciones con matrices en Matlab se dan de la siguiente manera:

- **Operaciones entre matrices**

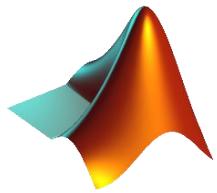
Simbolo	Operación
$+$	Suma de matrices $A + B \rightarrow A + B$
$-$	Resta de matrices $A - B \rightarrow A - B$
$*$	Multiplicación de un escalar con una matriz $cA \rightarrow c * A$ Multiplicación de matrices $AB \rightarrow A * B, \quad A = (a_{ij})_{n \times k} \quad B = (b_{ij})_{k \times m}$
$^\wedge$	Potenciación de una matriz $A^n \rightarrow A ^\wedge n, \quad A = (a_{ij})_{n \times n}$
\backslash	División a izquierda $A^{-1}B \rightarrow A \backslash B = inv(A) * B, \quad A = (a_{ij})_{n \times n}$ inversible
$/$	División a derecha $BA^{-1} \rightarrow B / A = B * inv(A), \quad A = (a_{ij})_{n \times n}$ inversible



```
>> A=[1 0 2; 0 5 1; 4 2 5]  
  
A =  
  
1 0 2  
0 5 1  
4 2 5  
  
>> B=[1 2 3; 4 5 6; 7 8 9]  
  
B =  
  
1 2 3  
4 5 6  
7 8 9
```

A continuación se muestra ejemplos para las operaciones usuales con matrices (Suma, Resta, Multiplicación y Potenciación).

<pre>>> A+B ans = 2 2 5 4 10 7 11 10 14 >> A-B ans = 0 -2 -1 -4 0 -5 -3 -6 -4</pre>	<pre>>> A*B ans = 15 18 21 27 33 39 47 58 69 >> A^2 ans = 9 4 12 4 27 10 24 20 35</pre>
--	--

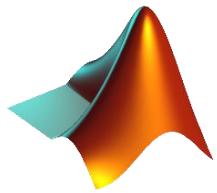


El sentido de división a izquierda y derecha, es de multiplicar por su inversa.

```
>> A\b  
  
ans =  
  
    1.8235    0.8235   -0.1765  
    0.8824    0.8824    0.8824  
   -0.4118    0.5882   1.5882  
  
>> B\A  
  
ans =  
  
    1.7059    0.4706   -0.1765  
    0.4706    0.6471    0.8824  
   -0.7647    0.8235   1.9412
```

- **Operaciones elemento a elemento**

símbolo	Operación
\cdot^*	Producto elemento a elemento $A \cdot^* B \rightarrow (a_{ij})_{m \times n} \cdot^* (b_{ij})_{m \times n} = (a_{ij} * b_{ij})_{m \times n}$
\cdot^\wedge	Potenciación elemento a elemento $A \cdot^\wedge B \rightarrow (a_{ij})_{m \times n} \cdot^\wedge (b_{ij})_{m \times n} = (a_{ij}^{b_{ij}})_{m \times n}$
$\cdot/$	Cociente elemento a elemento $A \cdot / B \rightarrow (a_{ij})_{m \times n} \cdot / (b_{ij})_{m \times n} = \left(\frac{a_{ij}}{b_{ij}} \right)_{m \times n}$
$\cdot\backslash$	Cociente elemento a elemento $A \cdot \backslash B \rightarrow (a_{ij})_{m \times n} \cdot \backslash (b_{ij})_{m \times n} = \left(\frac{b_{ij}}{a_{ij}} \right)_{m \times n}$



```
>> A=[1 2 3 ; 4 5 6]
```

```
A =
```

1	2	3
4	5	6

```
>> B=[6 5 4 ;3 2 1]
```

```
B =
```

6	5	4
3	2	1

```
>> A.*B
```

```
ans =
```

6	10	12
12	10	6

```
>> A.^B
```

```
ans =
```

1	32	81
64	25	6

Con el fin de poder apreciar con claridad la división a izquierda y derecha elemento a elemento, cambiaremos a 'format rat'.

```
>> format rat
>> A./B
```

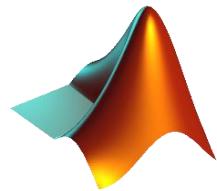
```
ans =
```

1/6	2/5	3/4
4/3	5/2	6

```
>> A.\B
```

```
ans =
```

6	5/2	4/3
3/4	2/5	1/6



Características de la matriz

Para conocer las características de una matriz, podemos utilizar los siguientes comandos.

Simbolo	Operación
[f c]=size(A)	Escrito de esta forma calcula la cantidad de filas 'f' y columnas 'c' que posee la matriz A.
det(A)	Calcula el determinante de la matriz A
rank(A)	Halla el rango de una matriz A.
trace(A)	Muestra la traza de la matriz A.

En general, definamos una matriz rectangular

```
>> A=[5 4 2; 8 3 7]

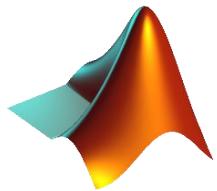
A =
      5     4     2
      8     3     7

>> [f c]=size(A)

f =
      2

c =
      3
```

Puesto que para el cálculo de la determinante se debe tener una matriz cuadrada, definamos una nueva matriz A.



```
>> A=[1 0 2; 0 5 1; 4 2 5]

A =

    1     0     2
    0     5     1
    4     2     5

>> det(A)

ans =

    -17

>> rank(A)

ans =

    3

>> trace(A)

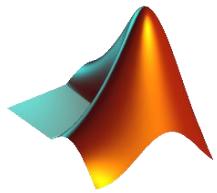
ans =

    11
```

Matrices especiales

Existen matrices que por su particularidad y debido a que son muy utilizados Matlab las tiene implementadas, con lo cual nos facilitara el trabajo con matrices. Veamos cómo llamarlas a ejecutarse.

Comando	Operación
eye(n)	Devuelve la matriz identidad de orden n.
zeros(n,m)	Devuelve la matriz nula de dimensiones nxm.
rand(n)	Devuelve una matriz aleatoria uniforme de orden n (o nxm), con valores entre 0 y 1.
randn(n)	Devuelve una matriz aleatoria normal de orden n (o nxm).
ones(n,m)	Devuelve una matriz de dimensiones nxm con



	todos sus elementos iguales a 1.
A'	Muestra la transpuesta de la matriz A.
Inv(A)	Devuelve la matriz inversa de una matriz A no singular (inversible).
diag(v)	Devuelve una matriz diagonal con los elementos del vector v.
diag(A)	Extrae la diagonal de la matriz A como un vector columna
Tril(A)	Devuelve la parte triangular inferior de una matriz A.
Triu(A)	Devuelve la parte triangular superior de una matriz A.

```
>> eye(3)          >> rand(3)

ans =              ans =

    1   0   0       0.5472   0.2575   0.8143
    0   1   0       0.1386   0.8407   0.2435
    0   0   1       0.1493   0.2543   0.9293

>> zeros(2,3)      >> rand(3,4)

ans =              ans =

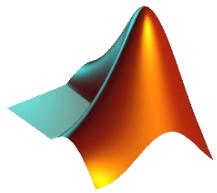
    0   0   0       0.3500   0.6160   0.8308   0.9172
    0   0   0       0.1966   0.4733   0.5853   0.2858
                           0.2511   0.3517   0.5497   0.7572

>> ones(3,2)        >> randn(2,3)

ans =              ans =

    1   1           0.8351   0.2157   -1.1480
    1   1           -0.2437  -1.1658   0.1049
    1   1
```

A' es la transpuesta de
la matriz A, A=(A)'

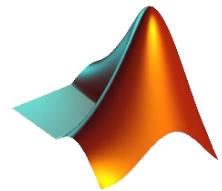


```
>> A=[1 0 2;0 5 1;4 2 5]  
  
A =  
  
1 0 2  
0 5 1  
4 2 5  
  
>> A'  
  
ans =  
  
1 0 4  
0 5 2  
2 1 5  
  
>> inv(A)  
  
ans =  
  
-1.3529 -0.2353 0.5882  
-0.2353 0.1765 0.0588  
1.1765 0.1176 -0.2941
```

```
>> diag([1 3 5 7])  
  
ans =  
  
1 0 0 0  
0 3 0 0  
0 0 5 0  
0 0 0 7  
  
>> diag(A)  
  
ans =  
  
1  
5  
5
```

```
>> tril(A)  
  
ans =  
  
1 0 0  
0 5 0  
4 2 5  
  
>> triu(A)  
  
ans =  
  
1 0 2  
0 5 1  
0 0 5
```

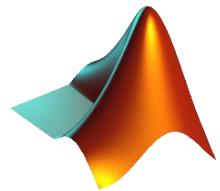
inv(A) se aplica a una matriz regular, (también llamado no singular, o inversible).



Funciones elementales que admiten como argumento una matriz compleja Z

- Trigonometrías

Función	Significado
sin(z)	Función seno
sinh(Z)	Función seno hiperbólico
asin(Z)	Función arcoseno
asinh(Z)	Función arcoseno hiperbólico
cos(Z)	Función coseno
cosh(Z)	Función coseno hiperbólico
acos(Z)	Función arcocoseno
acosh(Z)	Función arcocoseno hiperbólico
tan(Z)	Función tangente
tanh(Z)	Función tangente hiperbólica
atan(Z)	Función arcotangente
atanh(Z)	Función arcotangente hiperbólica
cot(Z)	Función cotangente
coth(Z)	Función cotangente hiperbólica
acot(Z)	Función arcocotangente
acoth(Z)	Función arcocotangente hiperbólica
sec(Z)	Función secante
sech(Z)	Función secante hiperbólica
asec(Z)	Función arcosecante
asech(Z)	Función arcosecante hiperbólica
csc(Z)	Función cosecante
csch(Z)	Función cosecante hiperbólica
acsc(Z)	Función arcocosecante
acsch(Z)	Función arcocosecante hiperbólica



Dada

$$Z = \begin{pmatrix} 1+i & 3+2i \\ 5+2i & -4+i \end{pmatrix}$$

```
>> z=[1+i 3+2*i; 5+2*i -4+i]

z =
1.0000 + 1.0000i 3.0000 + 2.0000i
5.0000 + 2.0000i -4.0000 + 1.0000i

>> sin(z)

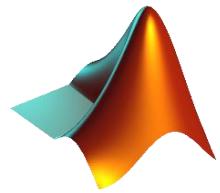
ans =
1.2985 + 0.6350i 0.5309 - 3.5906i
-3.6077 + 1.0288i 1.1678 - 0.7682i

>> sec(z)

ans =
0.4983 + 0.5911i -0.2635 + 0.0362i
0.0806 - 0.2628i -0.5578 + 0.4918i
```

- **Exponenciales**

Función	Significado
exp(z)	Función exponencial de base e
log(Z)	Función logaritmo neperiano
log10(Z)	Función logaritmo decimal
sqrt(Z)	Función raíz cuadrada



```
>> exp(z)

ans =

    1.0e+02 *

    0.0147 + 0.0229i  -0.0836 + 0.1826i
   -0.6176 + 1.3495i  0.0001 + 0.0002i

>> log(z)

ans =

    0.3466 + 0.7854i  1.2825 + 0.5880i
    1.6836 + 0.3805i  1.4166 + 2.8966i
```

- **Complejas**

Función	Significado
abs(z)	Modulo o valor absoluto
angle(Z)	Argumento
conj(Z)	Complejo conjugado
imag(Z)	Parte imaginaria
real(Z)	Parte real

```
>> abs(z)

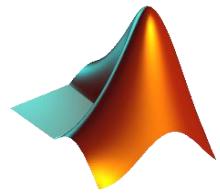
ans =

    1.4142    3.6056
    5.3852    4.1231

>> angle(z)

ans =

    0.7854    0.5880
    0.3805    2.8966
```



```
>> conj(z)

ans =

    1.0000 - 1.0000i  3.0000 - 2.0000i
    5.0000 - 2.0000i -4.0000 - 1.0000i

>> imag(z)

ans =

    1      2
    2      1

>> real(z)

ans =

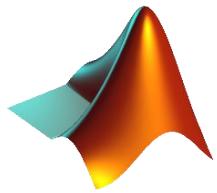
    1      3
    5     -4
```

• Numéricas

Función	Significado
fix(z)	Elimina las partes decimales
floor(Z)	Redondea los decimales al menor entero más cercano
ceil(Z)	Redondea los decimales al mayor entero más cercano
round(Z)	Efectúa el redondeo común de decimales

Modifiquemos nuestra matriz z, con el fin de entender esta tabla.

$$Z = \begin{pmatrix} 0.8 + 0.2i & 1.9 + 3.2i \\ 5.8 + 2.7i & -4.6 + 1.1i \end{pmatrix}$$



```
>> z=[0.8+0.2*i, 1.9+3.2*i; 5.8+2.7*i, -4.6+1.1*i]

z =
0.8000 + 0.2000i 1.9000 + 3.2000i
5.8000 + 2.7000i -4.6000 + 1.1000i

>> fix(z)

ans =
0.0000 + 0.0000i 1.0000 + 3.0000i
5.0000 + 2.0000i -4.0000 + 1.0000i

>> floor(z)

ans =
0.0000 + 0.0000i 1.0000 + 3.0000i
5.0000 + 2.0000i -5.0000 + 1.0000i

>> ceil(z)

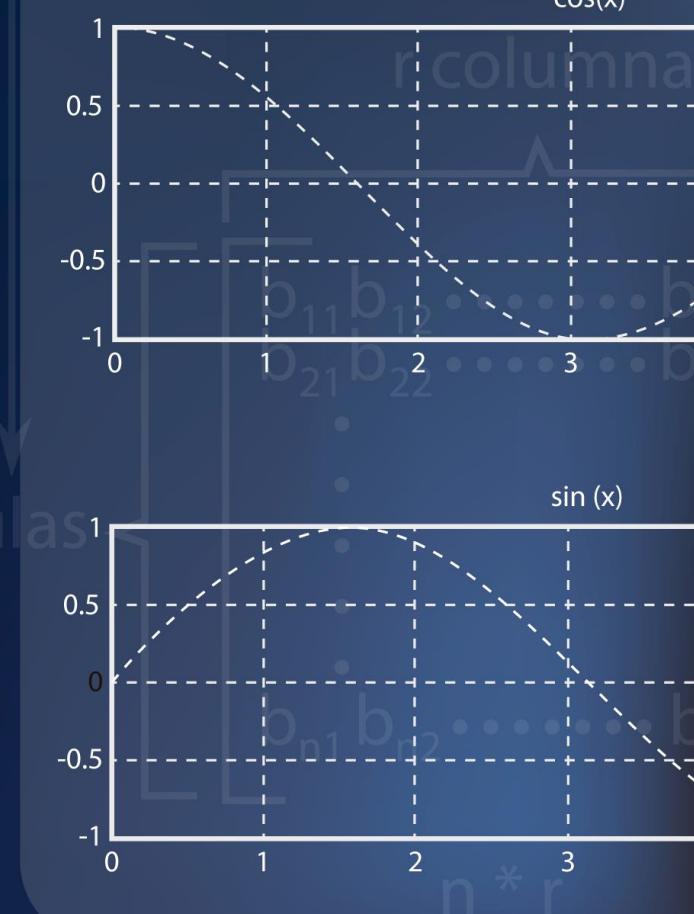
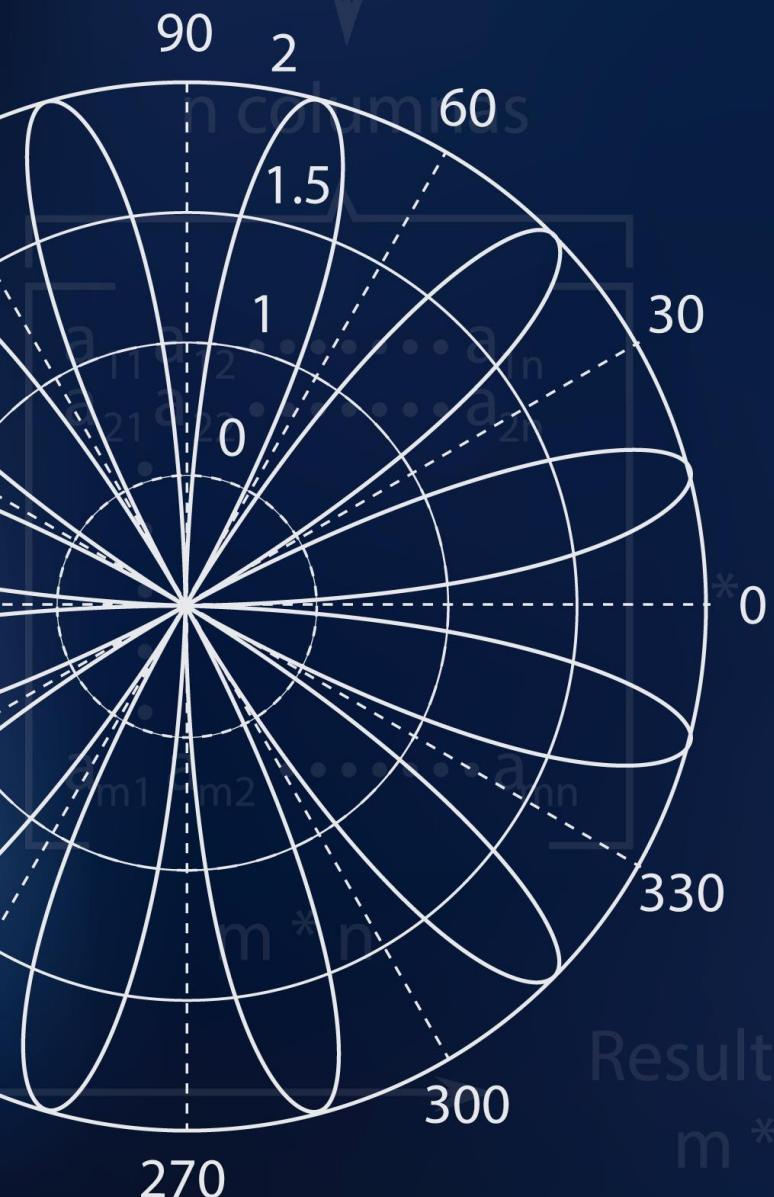
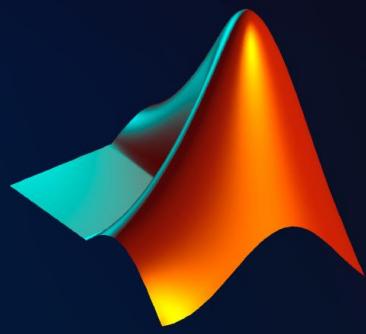
ans =
1.0000 + 1.0000i 2.0000 + 4.0000i
6.0000 + 3.0000i -4.0000 + 2.0000i

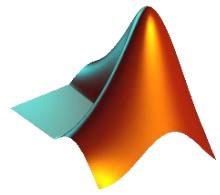
>> round(z)

ans =
1.0000 + 0.0000i 2.0000 + 3.0000i
6.0000 + 3.0000i -5.0000 + 1.0000i
```

Curso MATLAB

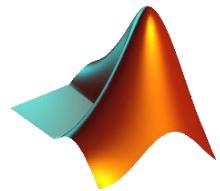
Básico





Contenido

Solución de sistemas lineales	3
Operaciones elementales fila y columna de matrices.....	8
• Operaciones fila.....	8
• Operaciones columna	9
Autovalores y autovectores de una matriz	10
Normas de una matriz	13



Sesión 7: Sistemas de ecuaciones lineales

Solución de sistemas lineales

Ahora vamos a estudiar los recursos de MatLab para resolver sistemas de ecuaciones lineales. Consideremos $Ax = B$

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \cdot \quad \cdot \quad \cdot \quad \quad \quad \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \quad \quad \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \quad \quad \quad \cdot \quad \cdot \quad \cdot \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{array} \right.$$

Si A es una matriz no singular, entonces $A \setminus B$ y B / A corresponden respectivamente a la multiplicación por izquierda y derecha de B por A^{-1} .

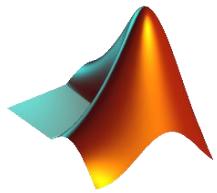
División Izquierda:

$x = A \setminus B$ Soluciona la ecuación matricial $Ax = B$

División derecha:

$x = B / A$ Soluciona la ecuación matricial $xA = B$

La función `x=linsolve(A,B)` de Matlab soluciona el sistema $Ax = B$



Ejemplo: Resolver

$$\begin{cases} 2x_1 + 3x_2 - 4x_3 = 3 \\ x_1 - x_2 + x_3 = -0.5 \\ 4x_1 - 7x_2 + 14x_3 = 2 \end{cases}$$

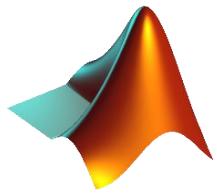
$$Ax = b$$

$$\begin{aligned} x &= \text{inv}(A) * b \\ &= A \setminus b \end{aligned}$$

En Matlab se resuelve del siguiente modo:

```
>> A=[2 3 -4;1 -1 1; 4 -7 14]  
  
A =  
  
    2     3     -4  
    1     -1      1  
    4     -7     14  
  
>> b=[3;-0.5;2]  
  
b =  
  
    3.0000  
   -0.5000  
    2.0000  
  
>> x=A\b  
  
x =  
  
    0.5000  
    2.0000  
    1.0000  
  
>> x=linsolve(A,b)  
  
x =  
  
    0.5000  
    2.0000  
    1.0000
```

Ambas formas
resuelven el
sistema $Ax = B$



Si desconocemos si una matriz es regular, como un recurso antes de aplicar el paso anterior podemos apoyarnos con su determinante.

Recordemos:

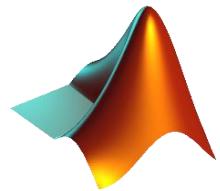
Una matriz es inversible, si y solo si, su determinante es distinto de cero.

Para nuestro ejemplo, $\det(A) \neq 0$

```
>> A=[2 3 -4;1 -1 1;4 -7 14]  
  
A =  
  
    2      3      -4  
    1     -1       1  
    4     -7      14  
  
>> det (A)  
  
ans =  
  
-32.0000
```

Otra alternativa de resolver el ejemplo anterior es usando el comando '**rref**' que requiere la matriz ampliada, para producir la forma escalonada reducida por filas de A usando eliminación de Gauss-Jordan con pivoteo parcial.

Veamos, a continuación:



```
>> Ab=[A b]

Ab =
2.0000    3.0000   -4.0000    3.0000
1.0000   -1.0000    1.0000   -0.5000
4.0000   -7.0000   14.0000    2.0000

>> rref(Ab)

ans =
1.0000         0         0    0.5000
0    1.0000         0    2.0000
0         0    1.0000    1.0000
```

De donde se observa que las soluciones corresponden a la última columna.

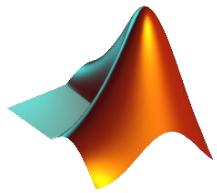
Por otro lado, la división matricial a izquierda obtiene la misma solución pero el método de resolución está basado en la factorización LU que es una modificación de la eliminación gaussiana.

Cuando usamos $x = A \setminus B$, internamente Matlab la resuelve de la siguiente forma.

- 1.- Calcula una matriz triangular inferior L, una matriz triangular superior U y una matriz de permutación P, tales que $PA=PU$. Donde P es la matriz identidad con sus filas cambiadas de orden.

- 2.- Resuelve $Ly=Pb$

- 3.- por último, se resuelve $Ux=y$



Por lo tanto, en Matlab será equivalente utilizar:

```
>> x = A \ b
```

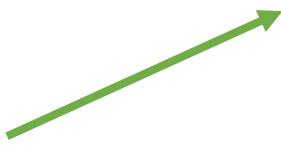
Que utilizar el siguiente proceso:

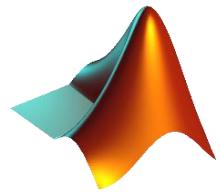
```
>> [L,U,P]=lu(A); %calcula las matrices L,U,P  
>> B=P*b;  
>> y=L\B;  
>> x=U\y
```

Verifiquemos,

```
>> [L,U,P]=lu(A)  
  
L =  
  
    1.0000      0      0  
    0.5000    1.0000      0  
    0.2500    0.1154    1.0000  
  
U =  
  
    4.0000   -7.0000   14.0000  
        0     6.5000  -11.0000  
        0         0   -1.2308  
  
P =  
  
    0      0      1  
    1      0      0  
    0      1      0  
  
>> B=P*b  
  
B =  
  
    2.0000  
    3.0000  
   -0.5000  
  
>> y=L\B  
  
y =  
  
    2.0000  
    2.0000  
   -1.2308  
  
>> x=U\y  
  
x =  
  
    0.5000  
    2.0000  
    1.0000
```

Desarrollo de la
solución $x = A \ B$





Operaciones elementales fila y columna de matrices

- **Operaciones fila**

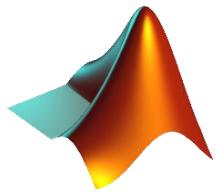
Sea,

$$A = \begin{pmatrix} 4 & 3 & 8 & 7 \\ 1 & 9 & 0 & 2 \\ -2 & 5 & 3 & 6 \end{pmatrix}$$

1° Multiplicar una fila por un escalar.

2° Intercambio de filas.

3° Sumar una fila a otra fila por un escalar.



```
>> A=[4 3 8 7; 1 9 0 2; -2 5 3 6]  
  
A =  
  
 4     3     8     7  
 1     9     0     2  
 -2     5     3     6  
  
>> A(1,:)=2*A(1,:)  
  
A =  
  
 8     6    16    14  
 1     9     0     2  
 -2     5     3     6  
  
>> aux=A(2,:); A(2,:)=A(1,:); A(1,:)=aux; A  
  
A =  
  
 1     9     0     2  
 8     6    16    14  
 -2     5     3     6  
  
>> A(2,:)=A(2,:)-3*A(1,:)
```

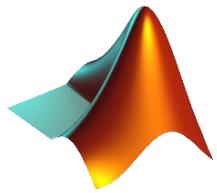
← ← ←

```
A =  
  
 1     9     0     2  
 5    -21    16     8  
 -2     5     3     6
```

- **Operaciones columna**

Sea,

$$B = \begin{pmatrix} -2 & 5 & 0 & 9 \\ 7 & 1 & -4 & 2 \\ 3 & 6 & 2 & 8 \end{pmatrix}$$



```
>> B=[-2 5 0 9; 7 1 -4 2; 3 6 2 8]  
  
B =  
  
-2 5 0 9  
7 1 -4 2  
3 6 2 8  
  
>> B(:,3)=0.5*B(:,3)  
  
B =  
  
-2 5 0 9  
7 1 -2 2  
3 6 1 8  
  
>> aux=B(:,1); B(:,1)=B(:,4); B(:,4)=aux; B  
  
B =  
  
9 5 0 -2  
2 1 -2 7  
8 6 1 3  
  
>> B(:,1)=B(:,1)-5*B(:,3)  
  
B =  
  
9 5 0 -2  
12 1 -2 7  
3 6 1 3
```

1º Multiplicar una columna por un escalar.

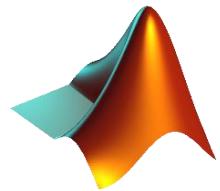
2º Intercambio de columnas.

3º Sumar una columna a otra columna por un escalar.

Autovalores y autovectores de una matriz

Recordemos algunos conceptos:

- 1.- El polinomio de grado n , $P_A(\lambda) = \det(A - \lambda I)$ se denomina polinomio característico de A .



2.- Las raíces del polinomio característico de A , se denominan autovalores (valores propios, valores característicos, o eigenvalores). Es decir λ es autovalor de A , si $P_A(\lambda) = 0$; por el teorema fundamental del álgebra, la matriz A tiene n autovalores (las cuales pueden ser reales o complejas, contando con su multiplicidad).

3.- El conjunto de todos los autovalores de una matriz A , se llama espectro, y se denombra por.

$$\sigma(A) = \{\lambda_i \text{ autovalor de } A, i = 1, \dots, n\}$$

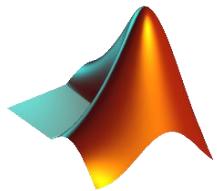
4.- El radiopectral de una matriz, se representa por

$$\rho(A) = \max_{1 \leq i \leq n} (|\lambda_i|), \quad \lambda_i \in \sigma(A)$$

Función	Significado
poly(A)	Polinomio característico de la matriz A .
jordan(A)	Forma canónica de Jordan de la matriz A .
eig(A)	autovalores de la matriz A .
[V,D]=eig(A)	Autovalores (diagonal de D) y Autovectores (columnas de V) de una matriz cuadrada A .

Esto quiere decir que el polinomio característico es,

$$P_A(x) = x^3 - 3x^2 + 4$$



```
>> A=[1 2 1; 0 -1 0; -1 1 3]

A =

    1     2     1
    0    -1     0
   -1     1     3

>> poly(A)

ans =

    1.0000    -3.0000    -0.0000    4.0000

>> jordan(A)

ans =

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

```
>> eig(A)

ans =

    2.0000
    2.0000
   -1.0000

>> [V,D]=eig(A)

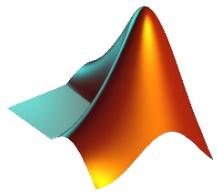
V =

    0.7071   -0.7071   -0.5793
            0          0    0.7448
    0.7071   -0.7071   -0.3310

D =

    2.0000         0         0
            0    2.0000         0
            0         0   -1.0000
```

Autovalores como un vector columna o como diagonal de una matriz.



Normas de una matriz

Norma	Significado
norm(A)	Norma 2 de la matriz A.
norm(A,1)	Norma 1 de la matriz A.
norm(A,inf)	Norma infinito de la matriz A.
norm(A,'fro')	Norma frobenius de la matriz A.

Norma - 1, es la máxima suma absoluta de las columnas de la matriz.

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^n |a_{ij}|$$

Norma - ∞ , es la máxima suma absoluta de las filas de la matriz.

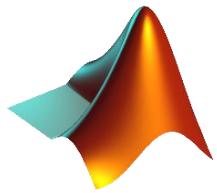
$$\|A\|_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^n |a_{ij}|$$

Norma - 2, es la raíz cuadrada del autovalor más grande de la matriz $A^t A$.

$$\|A\|_2 = \sqrt{\rho(A^t A)} = \sqrt{\rho(AA^t)}, \quad A^t \text{ es la transpuesta de } A.$$

Norma-frobenius, es la raíz cuadrada de la suma de los cuadrados de los elementos de la matriz A .

$$\|A\|_F = \sqrt{\sum_{i,j=1}^n |a_{ij}|^2} = \sqrt{\text{traza}(A^t A)} = \sqrt{\text{traza}(AA^t)}$$



Veamos los resultados obtenidos de las distintas normas mencionadas.

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

A =

    1     2     3
    4     5     6
    7     8     9

>> norm(A)

ans =

    16.8481

>> norm(A,1)

ans =

    18

>> norm(A,inf)

ans =

    24

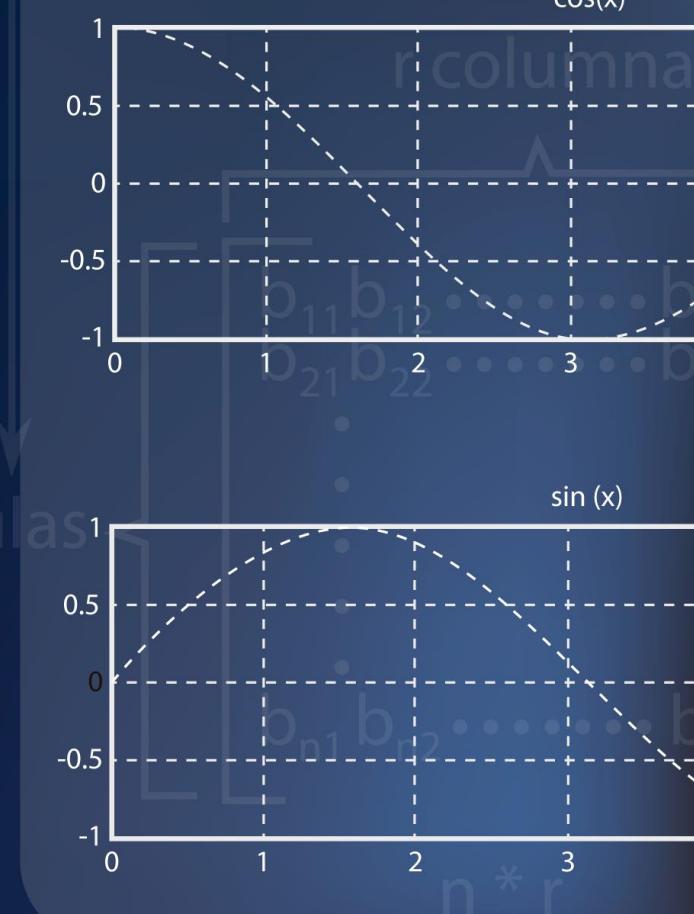
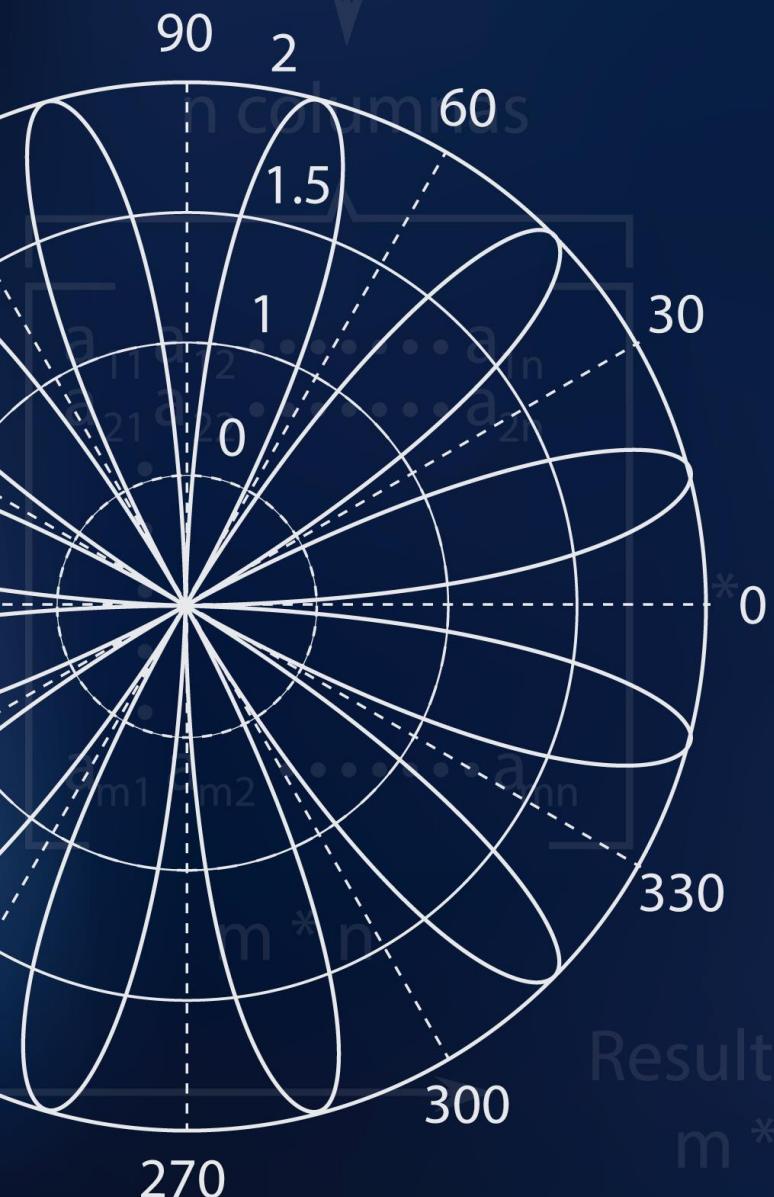
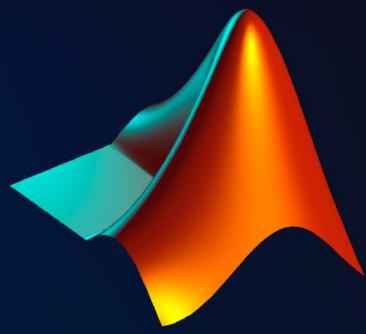
>> norm(A,'fro')

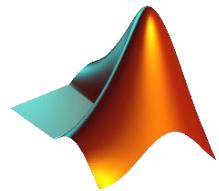
ans =

    16.8819
```

Curso MATLAB

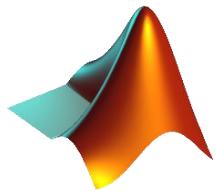
Básico





Contenido

Funciones graficas 2D.....	3
Curvas explicitas e implícitas	12
Subdivisión de ventanas	15
Control de los ejes	18
Títulos, etiquetas y colocación	20
Control de ventanas graficas	23
Gráficos en coordenadas polares	25



Sesión 8: Gráficas Bidimensionales

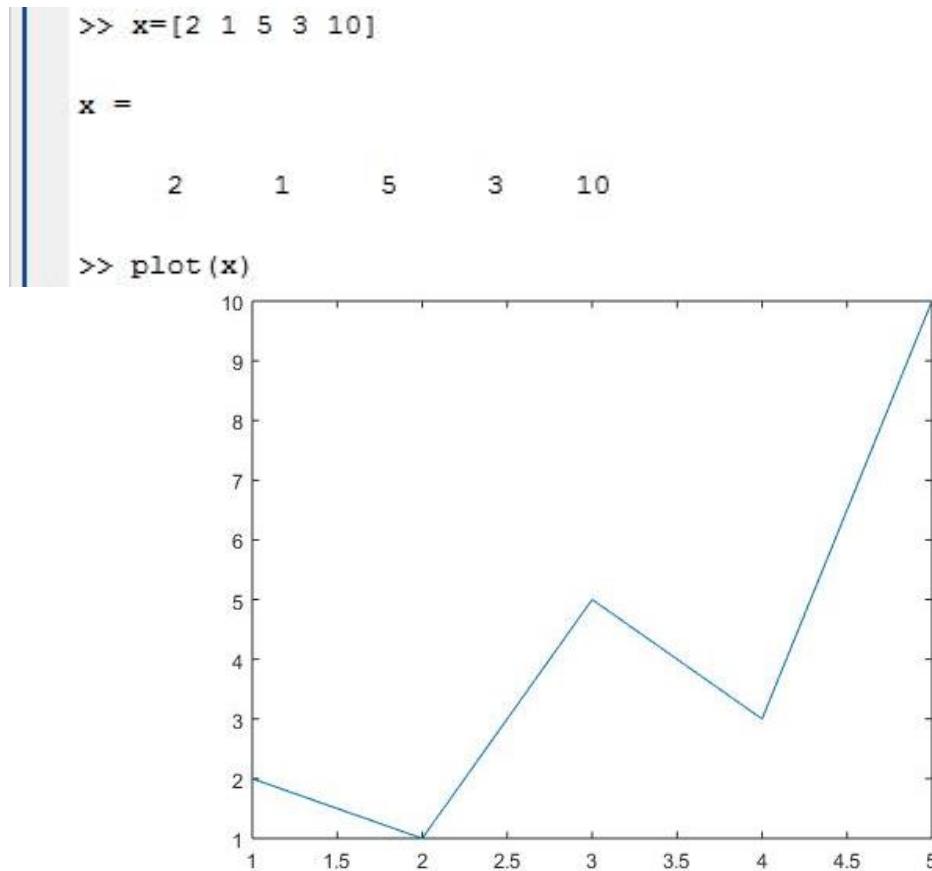
Funciones graficas 2D

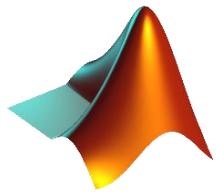
Matlab dispone de 5 funciones básicas para crear gráficos 2D. Estas funciones se diferencian principalmente por el tipo de escala que utilizan en el eje de las abscisas y de las ordenadas.

1.- **plot()**

Crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre los ejes.

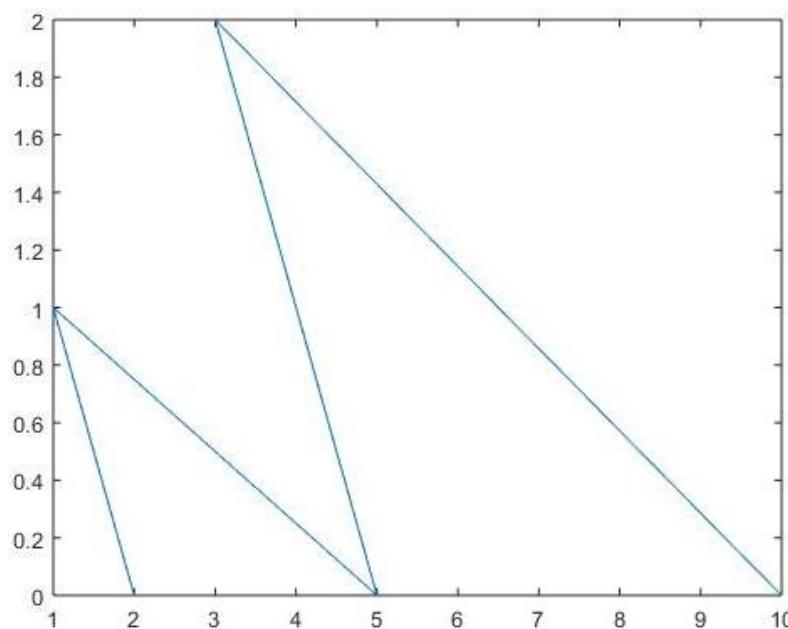
plot(X) si X es un vector dibuja los puntos (i, X_i) . Si X es una matriz hace lo mismo para cada columna de la matriz. Si es un vector complejo, dibuja los puntos $(\text{Re}(X_i), \text{Im}(X_i))$.





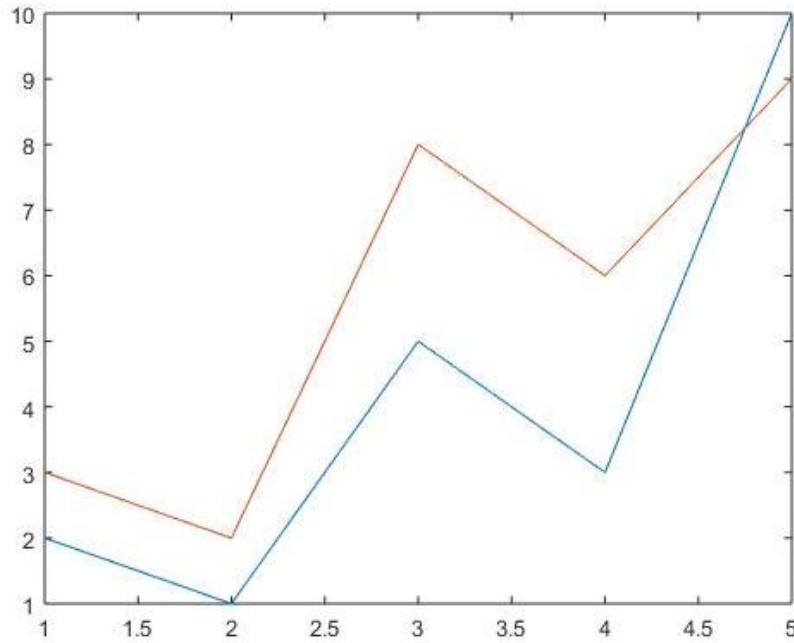
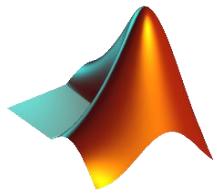
Ahora sobre un vector complejo

```
>> xc=[2 1+i 5 3+2i 10]  
  
xc =  
  
Columns 1 through 3  
  
2.0000 + 0.0000i 1.0000 + 1.0000i 5.0000 + 0.0000i  
  
Columns 4 through 5  
  
3.0000 + 2.0000i 10.0000 + 0.0000i  
  
>> plot(xc)
```



Veamos con una matriz

```
>> A=[2 3; 1 2; 5 8; 3 6; 10 9]  
  
A =  
  
2 3  
1 2  
5 8  
3 6  
10 9  
  
>> plot(A)
```



Para una matriz compleja es similar y el resultado es la gráfica de las columnas (vectores complejos) de la matriz.

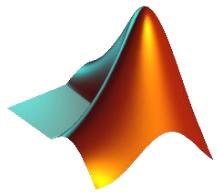
plot(X,Y)

Si X e Y son vectores de igual tamaño, dibuja Y versus X.

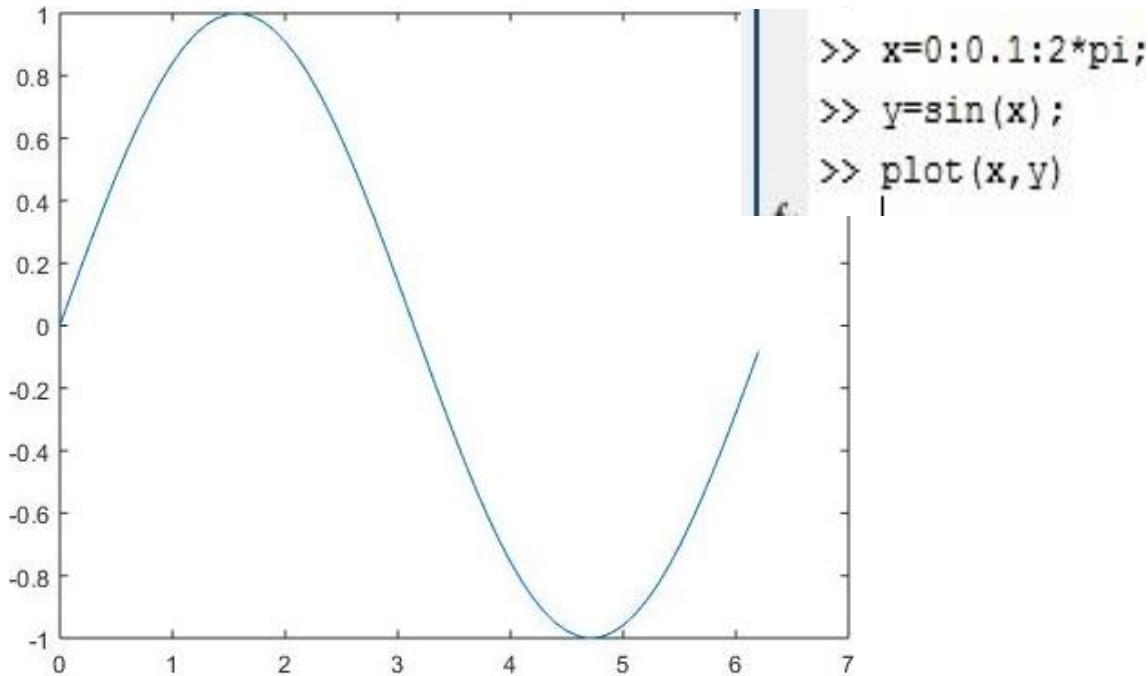
Si X e Y son matrices con las mismas dimensiones, dibuja columnas de Y frente a columnas de X.

Si uno de X o Y es un vector y la otra una matriz, alguna de las dimensiones de la matriz debe ser igual a la longitud del vector. Si el número de filas (o columnas) de la matriz es igual a la longitud del vector, entonces se dibuja cada columna (o fila) frente al vector. Si la matriz es cuadrada, grafica cada columna frente al vector.

Para valores complejos X e Y se ignoran las partes imaginarias.

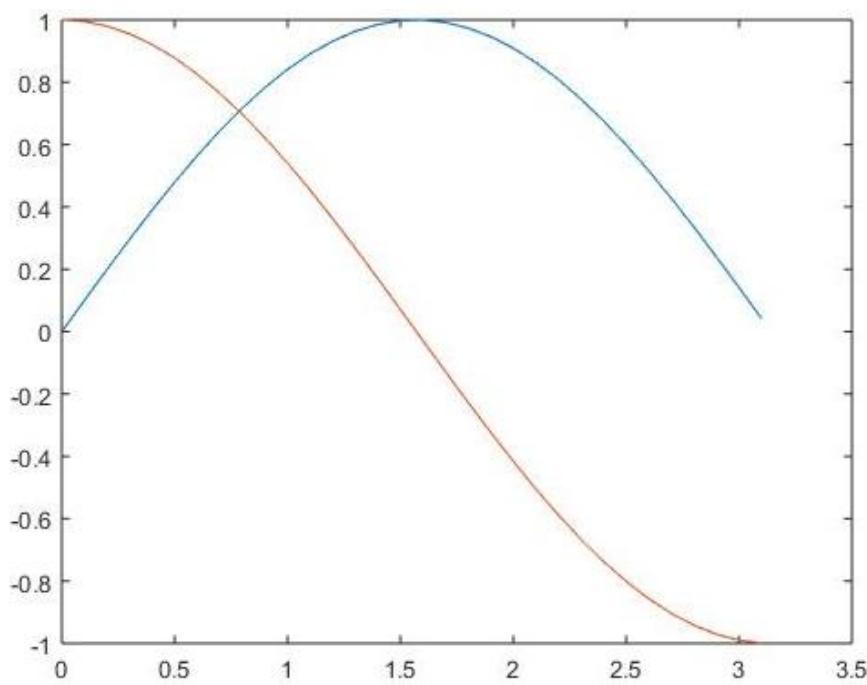


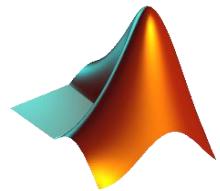
Veamos el caso X e Y vectores



Ahora un vector con una matriz

```
>> x=0:0.1:pi;
>> A=[sin(x) ; cos(x)];
>> plot(x,A)
```



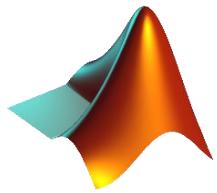


plot(X,Y,S) grafica de plot(X,Y) con las opciones definidas en S. usualmente S se compone de 3 caracteres entre comillas simples ('S'), el primero de los cuales fija el color de línea del gráfico, el segundo fija la etiqueta o marca en el nodo y el ultimo fija el carácter a usar en el graficado. Los colores, estilos de línea y marcas son respectivamente, los siguientes

Símbolo	Color
y	Yellow
m	Magenta
c	Cyan
r	Red
g	Green
b	Blue
w	White
k	Black

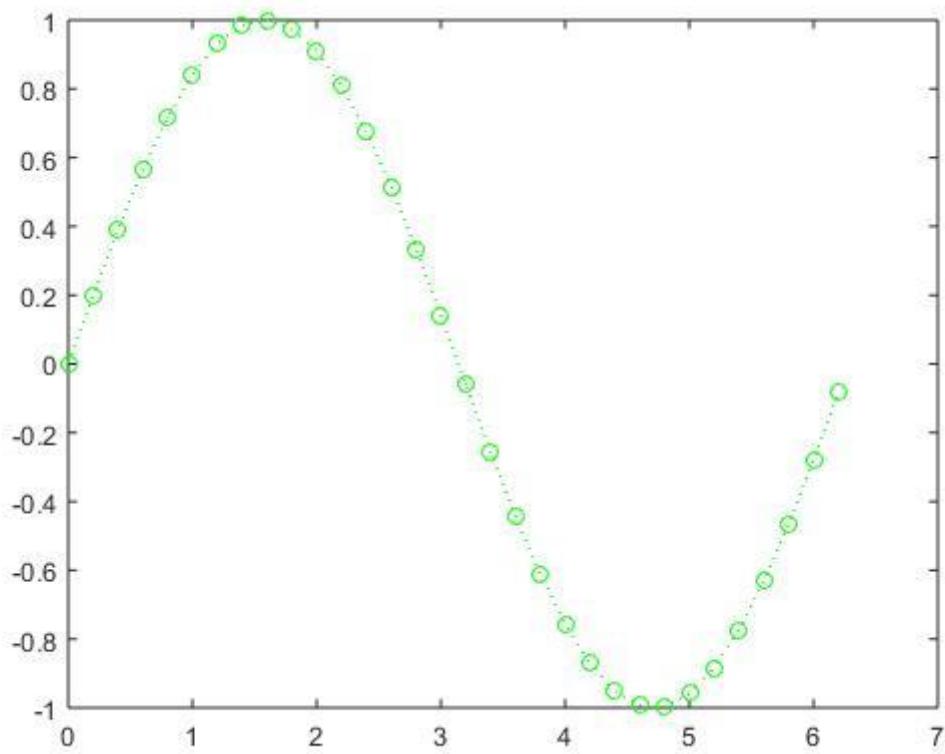
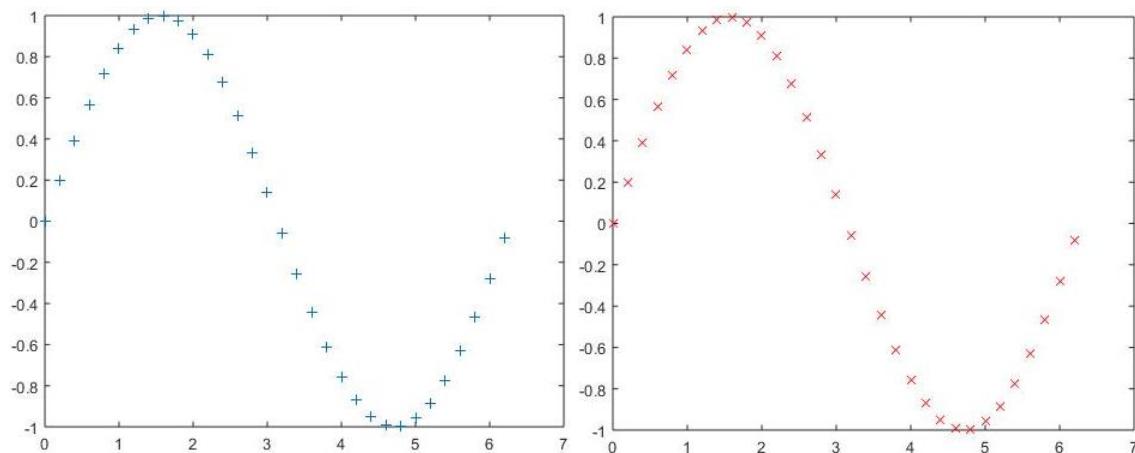
Símbolo	Estilo de línea
-	Línea continua
:	Línea a puntos
-.	Línea a barra-punto
--	Línea a trazos

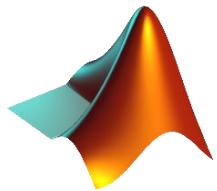
Símbolo	Marcadores
.	Puntos
o	Círculos
x	Marcas con x
+	Marcas con +
*	Marcas con *
s	Marcas cuadradas (square)
d	Marcas en diamantes (diamond)
^	Triangulo apuntando arriba
v	triangulo apuntando abajo
>	triangulo apuntando a la derecha
<	triangulo apuntando a la izquierda
p	estrella de 5 puntas
h	estrella de 6 puntas



Ejemplo:

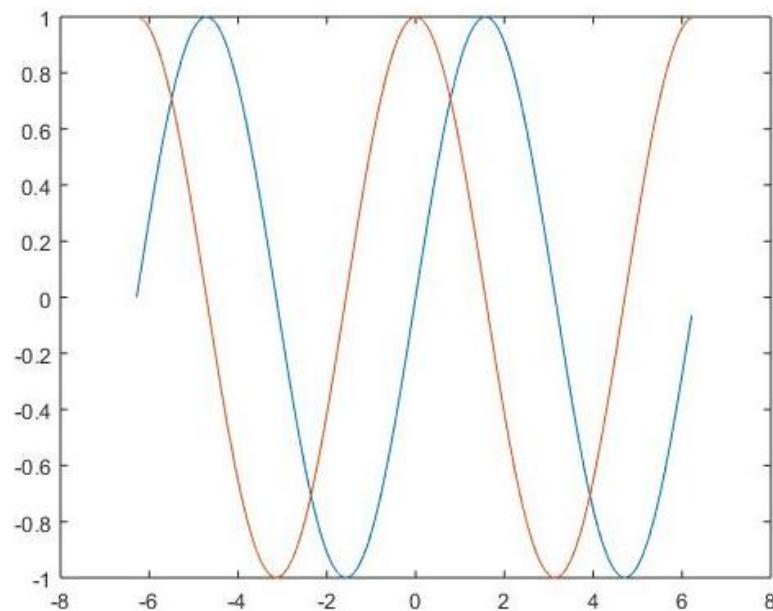
```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y,'+')
>> plot(x,y,'rx')
>> plot(x,y,'go:')
```



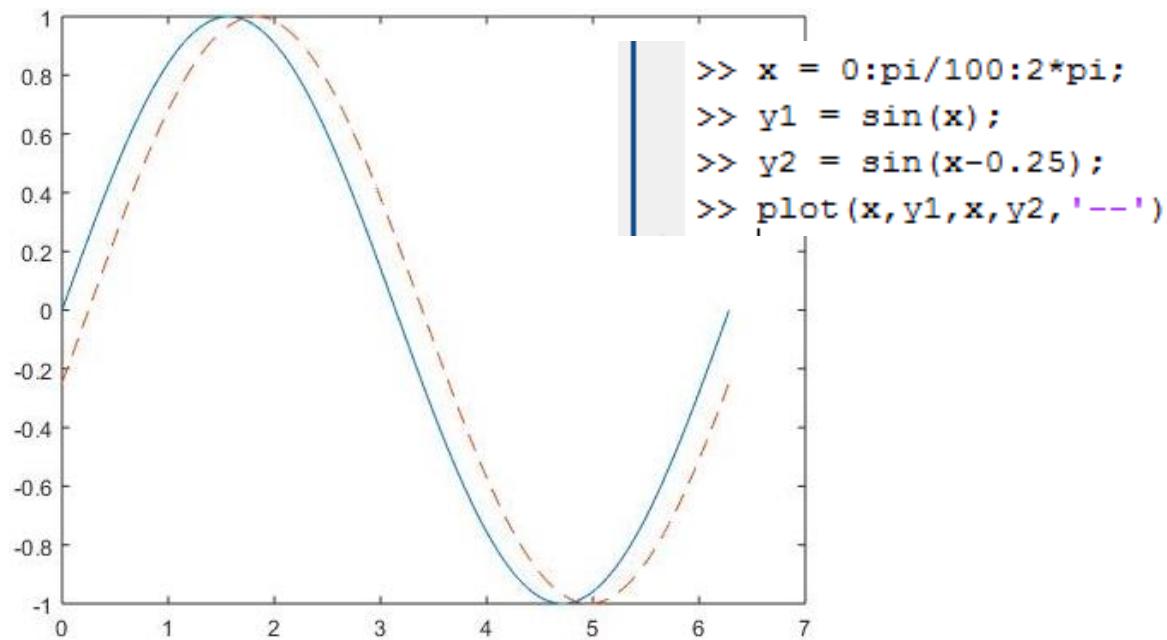


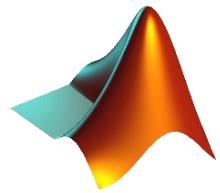
plot(X1,Y1,...,Xn,Yn) dibuja múltiples pares X, Y utilizando los mismos ejes para todas las líneas.

```
>> x=-2*pi:0.1:2*pi;  
>> y1=sin(x);  
>> y2=cos(x);  
>> plot(x,y1,x,y2)
```



plot(X1,Y1,S1,...,Xn,Yn,Sn) es similar a combinar Plot(X,Y,S) y Plot(X1,Y1,...,Xn,Yn).

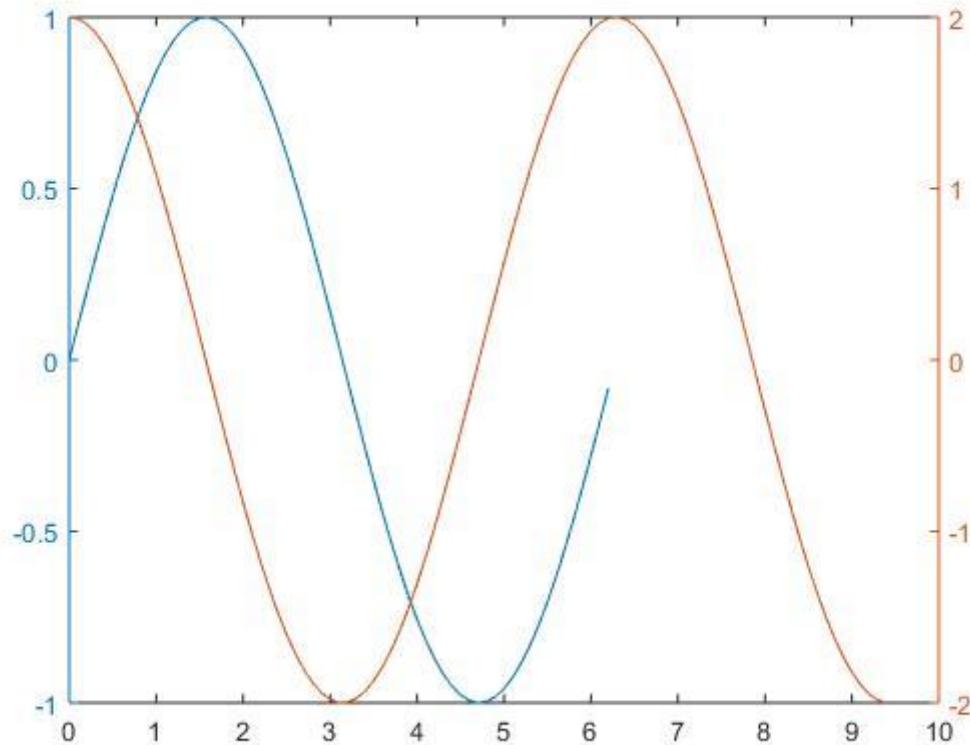




2.- `plotyy()` dibuja dos funciones con escalas diferentes para las ordenadas; una a la derecha y otra a la izquierda.

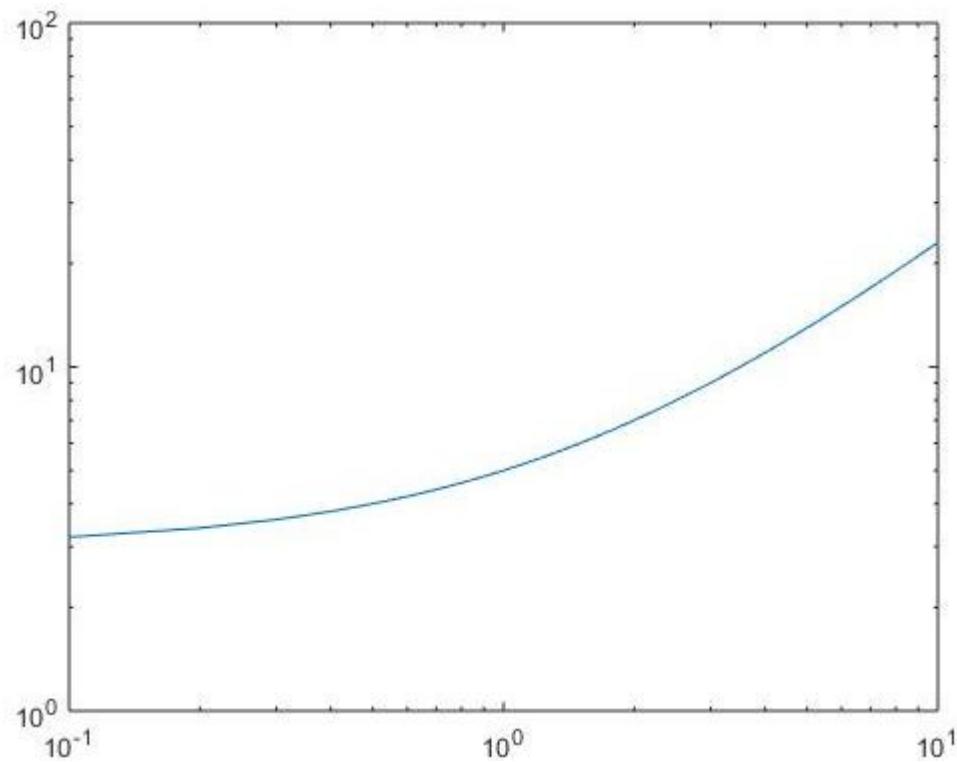
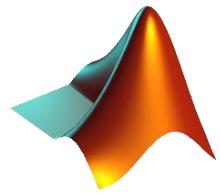
Ejemplo:

```
>> x1=0:0.1:2*pi;
>> y1=sin(x1);
>> x2=0:0.1:3*pi;
>> y2=2*cos(x2);
>> plotyy(x1,y1,x2,y2)
```

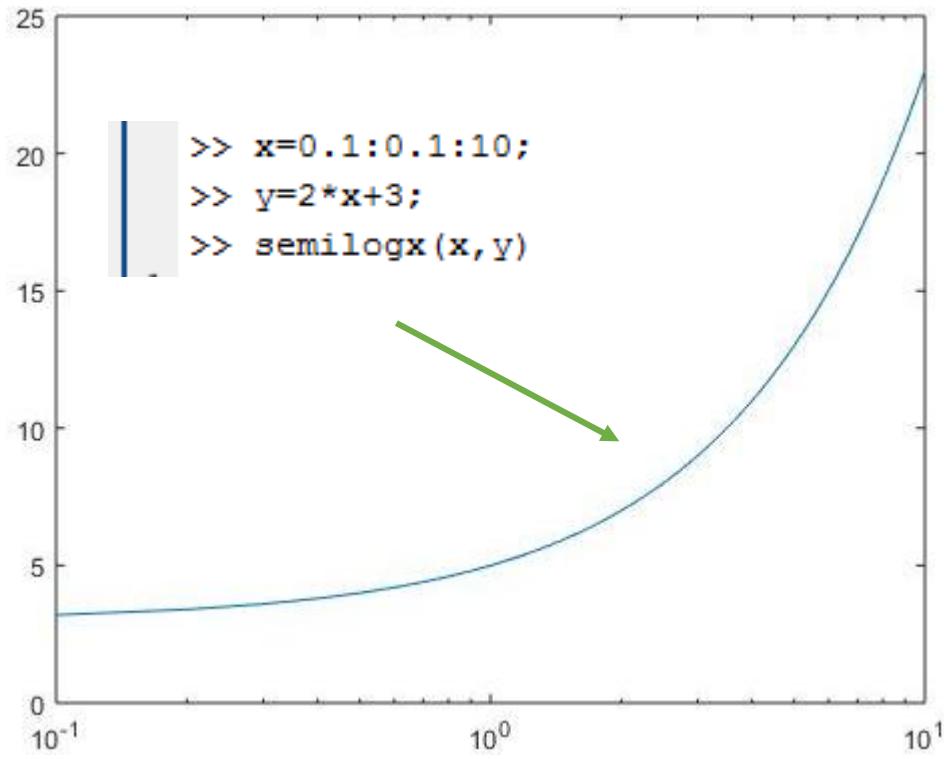


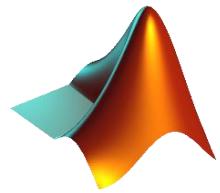
3.- `loglog()` realiza gráficos similares a `plot(X,Y)`, pero con escala logarítmica en los 2 ejes.

```
>> x=0.1:0.1:10;
>> y=2*x+3;
>> loglog(x,y)
```

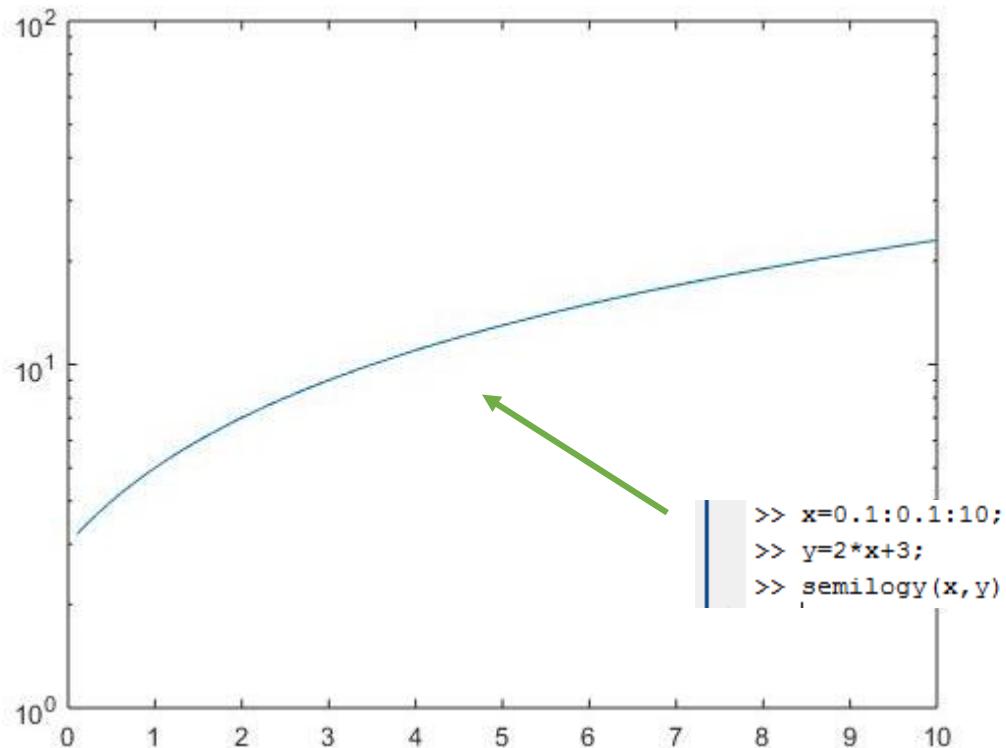


4.- `semilogx()` realiza gráficos similares a `plot(X,Y)`, pero con escala logarítmica en el eje X y escala normal en el eje Y.





5.- **semilogy()** realiza gráficos similares a plot(X,Y), pero con escala logarítmica en el eje Y y escala normal en el eje X.



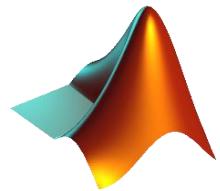
Curvas explícitas e implícitas

fplot()

fplot('f',[xmin, xmax]) grafica la función en el intervalo de variación de x dado.

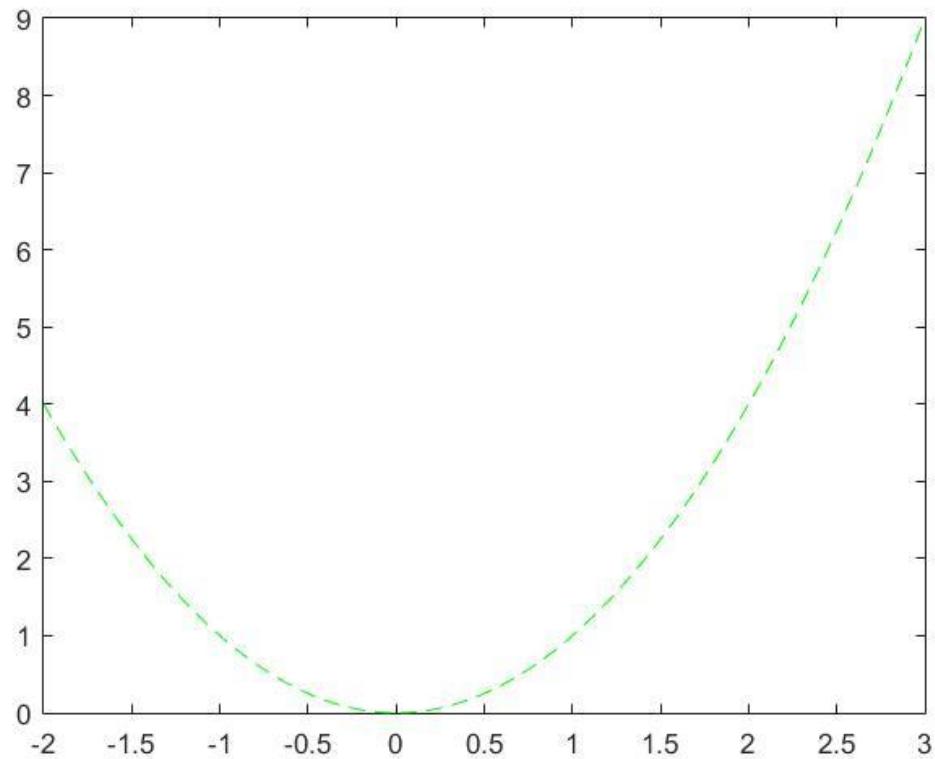
fplot('f',[xmin, xmax],S) similar a fplot('f',[xmin, xmax]) añadiendo las opciones de color y caracteres que se especifican en S.

fplot('f', [xmin, xmax,ymin,ymax],S) grafica la función en los intervalos de x e y dados, con las opciones de color y caracteres dadas por S.



Ejemplo:

```
>> fplot('x^2', [-2 3], 'g--')
```

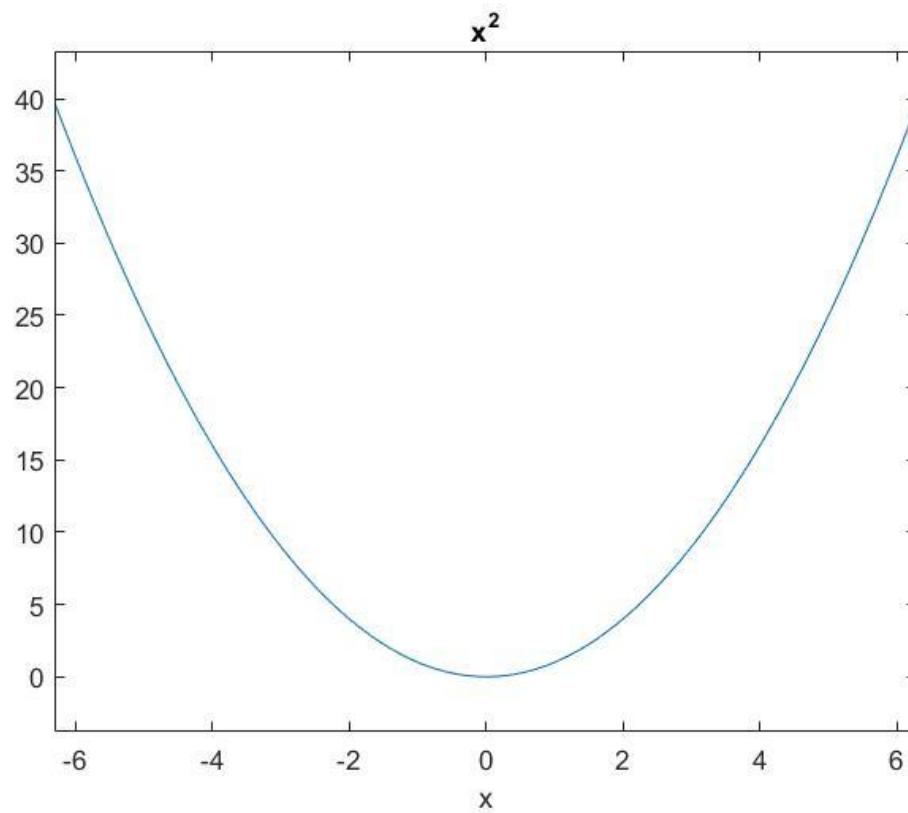
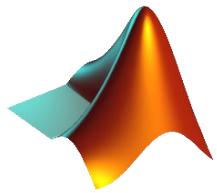


ezplot()

ezplot (fun) traza la expresión fun(x) sobre el dominio predeterminado $-2\pi < x < 2\pi$, donde fun(x) es una función explícita de una sola variable x.

ezplot (fun, [xmin, xmax]) representa la fun(x) sobre el dominio: $x_{\min} < x < x_{\max}$.

```
>> ezplot('x^2')
```



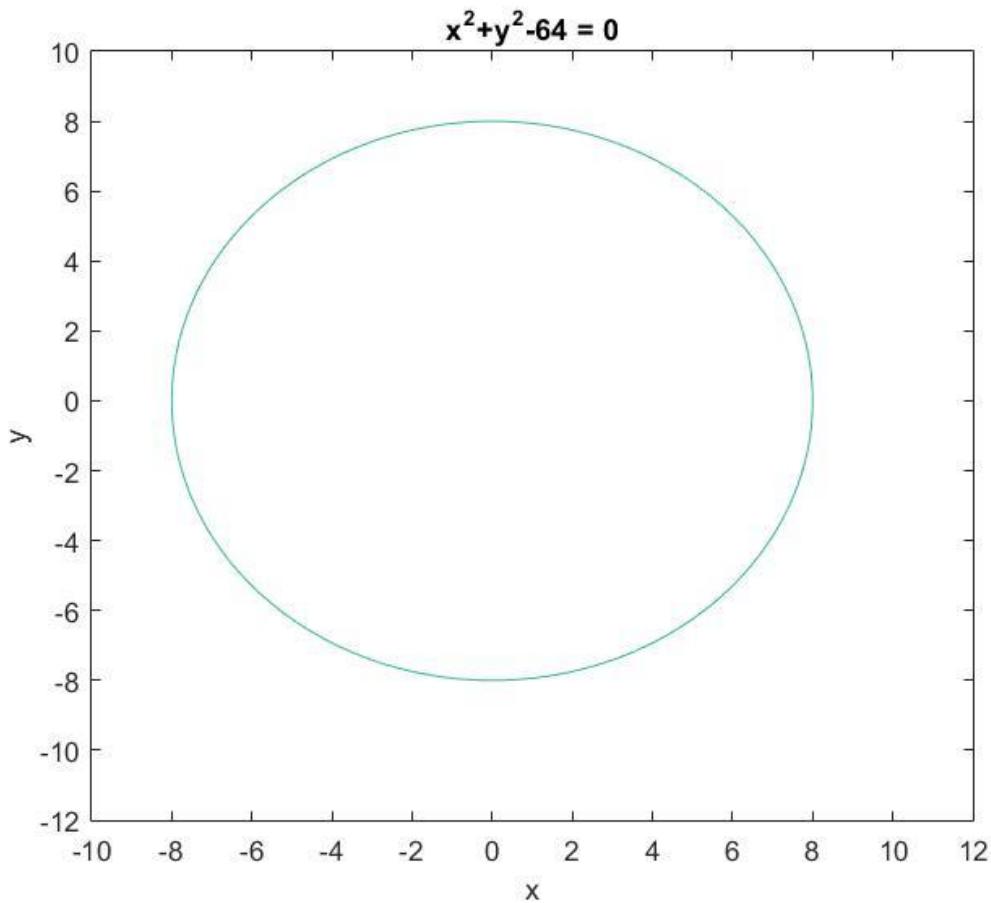
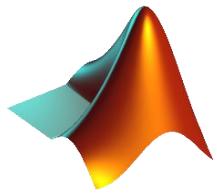
Para una función implícita, fun2 (x, y):

ezplot ('fun2') traza $\text{fun2}(x,y)=0$ sobre el dominio predeterminado $-2\pi < x < 2\pi$, $-2\pi < y < 2\pi$.

ezplot ('fun2', [a, b]) traza $\text{fun2}(x, y)=0$ sobre $a < x < b$ y $a < y < b$.

ezplot ('fun2', [xmin, xmax, ymin, ymax]) traza $\text{fun2}(x, y) = 0$ sobre $xmin < x < xmax$ y $ymin < y < ymax$.

```
>> ezplot('x^2+y^2-64', [-10 12 -12 10])
```

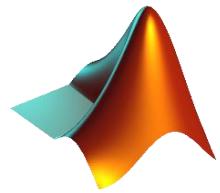


Subdivisión de ventanas

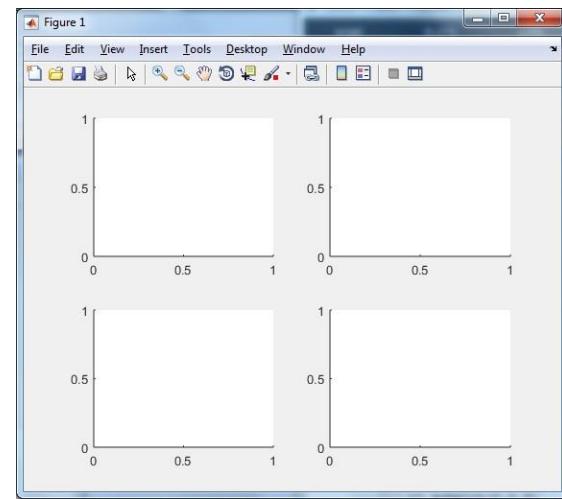
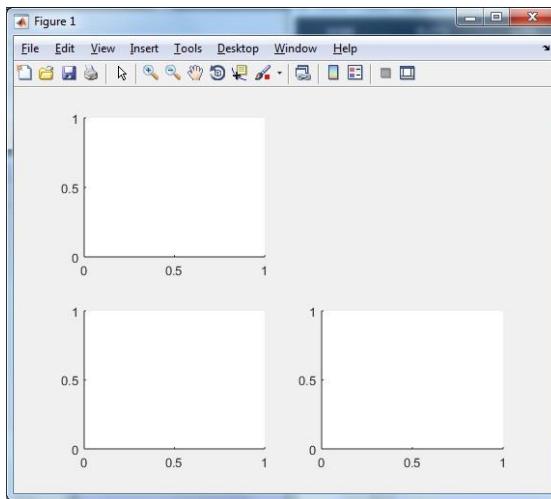
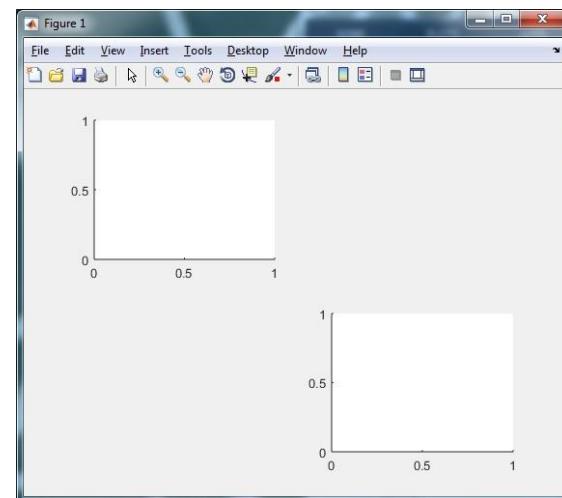
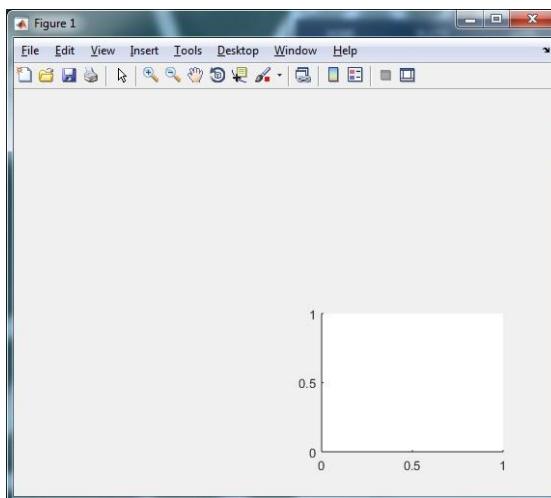
Una ventana grafica se puede dividir en **m** particiones horizontales y **n** verticales, con objeto de representar múltiples gráficos en ella. Cada una de estas subventanas tiene sus propios ejes, aunque otras propiedades son comunes a toda la figura. La forma general de este comando es:

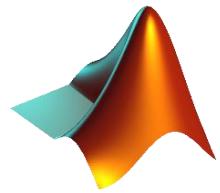
subplot(m,n,i)

Donde **m** y **n** son el número de subdivisiones en filas y columnas respectivamente, e **i** es la subdivisión que se convierte en activa. Las subdivisiones se enumeran consecutivamente empezando por la primera fila, siguiendo por las de la segunda, etc.



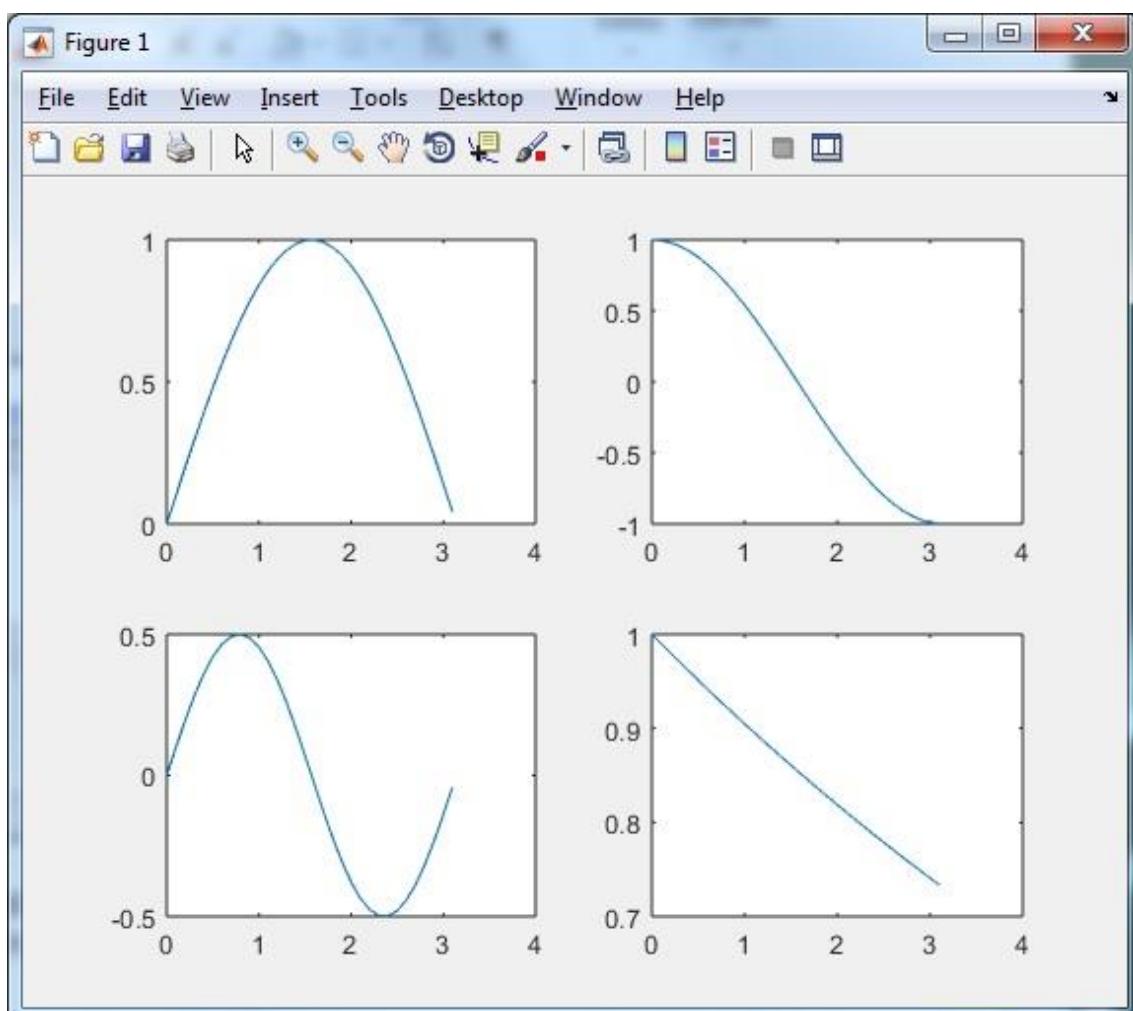
```
>> subplot(2,2,4)
>> subplot(2,2,1)
>> subplot(2,2,3)
>> subplot(2,2,2)
```

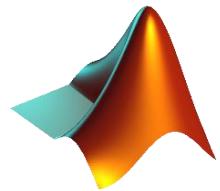




Vamos a realizar 4 gráficos distintos en una sola ventana grafica

```
>> x=0:0.1:pi;
>> y=sin(x);
>> z=cos(x);
>> v=y.*z;
>> w=exp(-0.1*x);
>> subplot(2,2,1), plot(x,y)
>> subplot(2,2,2), plot(x,z)
>> subplot(2,2,3), plot(x,v)
>> subplot(2,2,4), plot(x,w)
```





Control de los ejes

Matlab por defecto ajusta la escala de cada uno de los ejes de modo que varíe entre el mínimo y el máximo valor de los vectores a representar. En ciertas ocasiones es conveniente cambiar estas opciones por defecto llamadas modo 'auto', o modo automático. Con el comando básico **axis** podemos realizar distintos tipos de cambios:

axis([xmin, xmax, ymin, ymax]) define de modo explícito los valores máximo y mínimo según cada eje.

axis('auto') devuelve el escalado de los ejes al valor por defecto o automático.

v=axis devuelve un **vector v** con los valores **[xmin, xmax, ymin, ymax]**.

axis('ij') utiliza ejes de pantalla, con el origen en la esquina superior izquierda y el **eje j** en dirección vertical descendente.

axis('xy') utiliza ejes cartesianos normales, con el origen en la esquina inferior izquierda y el **eje y** vertical ascendente.

axis('image') la ventana tendrá las proporciones de la imagen que se desea representar en ella (por ejemplo la de una imagen bitmap que se deseé importar) y el escalado de los ejes será coherente con dicha imagen.

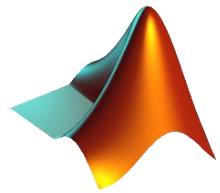
axis('equal') el escalado es igual en ambos ejes.

axis('square') la ventana será cuadrada.

axis('normal') elimina las restricciones introducidas por '**equal**' y '**square**'.

axis('off') elimina las etiquetas, los números y los ejes.

axis('on') restituye las etiquetas, los números y los ejes.

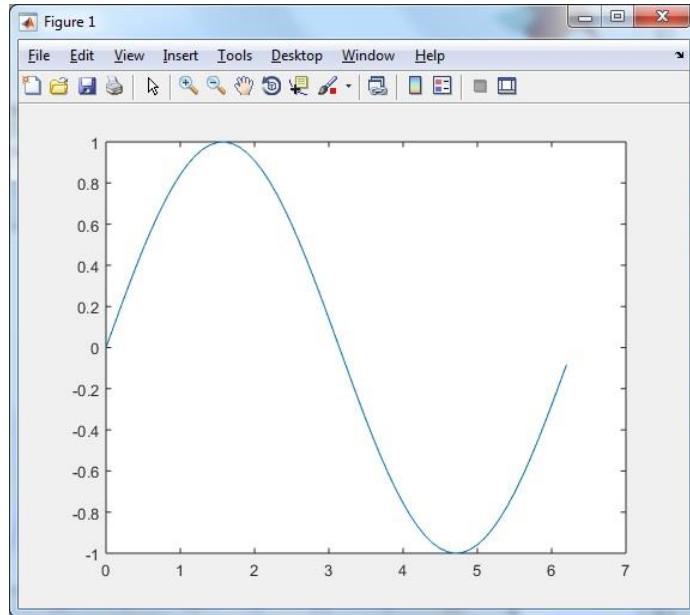


Veamos los distintos casos,

```
>> x=0:0.1:2*pi;
>> y=sin(x);
>> plot(x,y)
>> axis([-1, 7, -2, 2])
>> v=axis

v =
    -1      7     -2      2

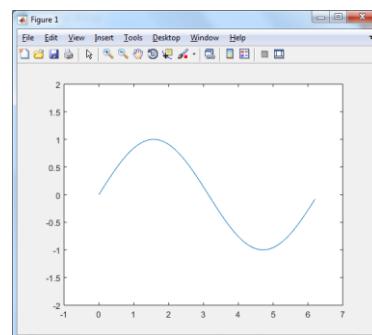
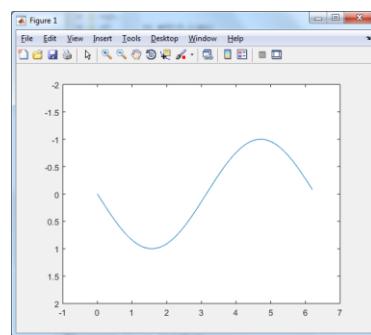
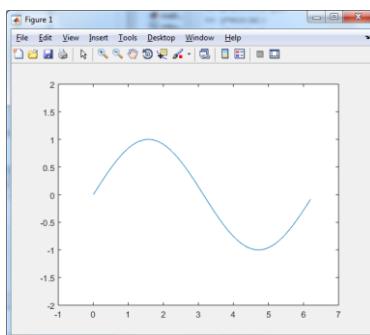
>> axis('ij')
>> axis('xy')
>> axis('image')
>> axis('equal')
>> axis('square')
>> axis('normal')
>> axis('off')
>> axis('on')
```



axis([-1 7 -2 2])

ij

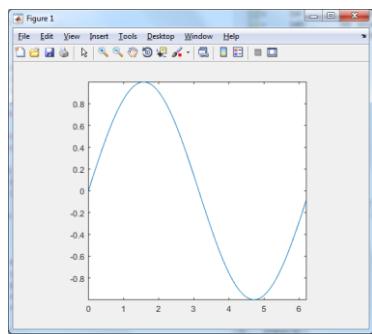
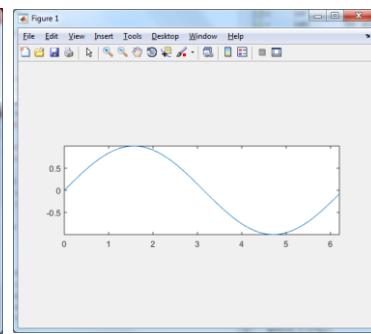
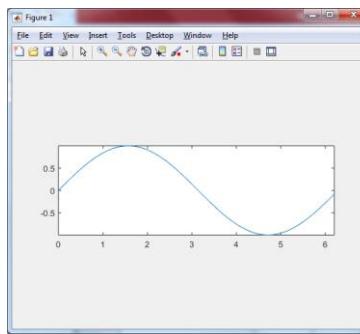
xy

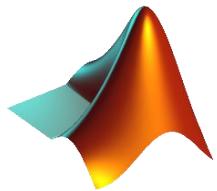


image

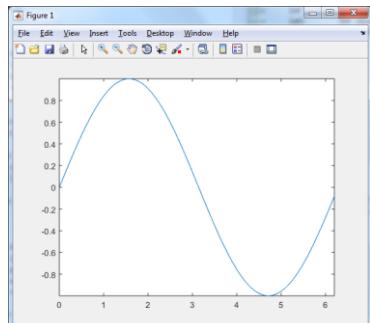
Equal

square

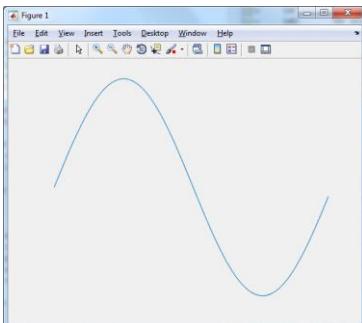




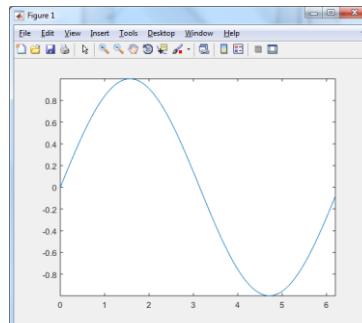
Normal



off



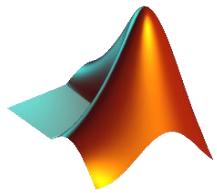
on



Títulos, etiquetas y colocación

Matlab permite manejar correctamente las anotaciones sobre los gráficos y ejes mediante colocación adecuada de títulos, etiquetas, leyendas, rejillas, etc. Los comandos más usuales son

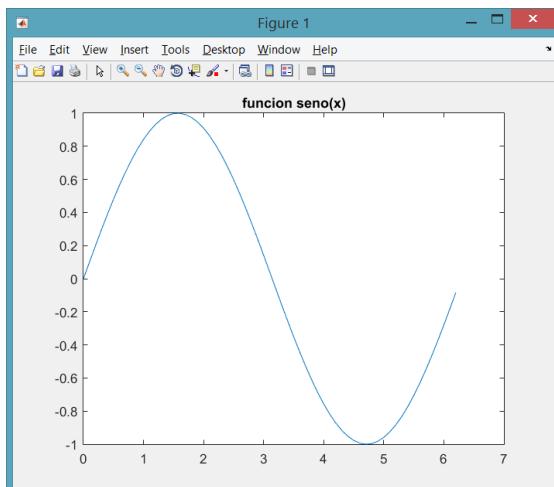
Comandos	Significado
title('texto')	Añade el texto como título del gráfico en la parte superior del mismo.
xlabel('texto') ylabel('texto')	Sitúa el texto al lado del eje x (o eje y), en gráficos 2D .
legend('cadena1', 'cadena2',...)	Sitúa las leyendas especificadas por las cadenas en n gráficos consecutivos.
legend('off')	Elimina las leyendas de los ejes actuales.
text(x,y,'texto')	Sitúa el texto en el punto (x,y) dentro del grafico 2D .
gtext('texto')	Permite situar el texto en un punto seleccionado con el ratón dentro de un gráfico 2D .
grid	Sitúa rejillas en los ejes de un gráfico; grid on mantiene activa las rejillas y grid off las elimina.
hold on	Activa la superposición de gráficos en los mismos ejes.
hold off	Desactiva la opción hold on .



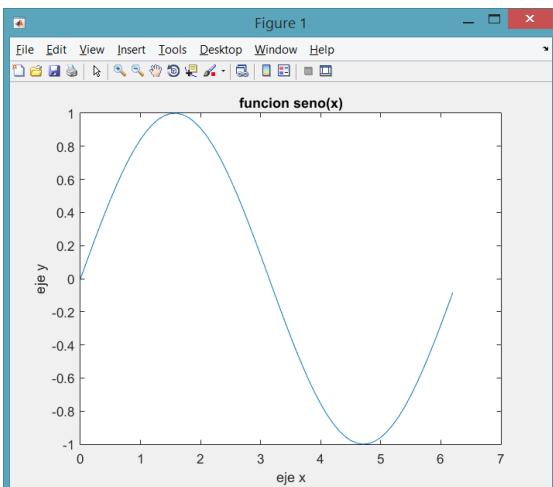
Ejemplo:

```
>> x=0:0.1:2*pi; y=sin(x); plot(x,y)
>> title('funcion seno(x)')
>> xlabel('eje x'), ylabel('eje y')
>> legend('seno(x)')
>> text(3,0,'texto')
>> gtext('texto2')
>> grid
>> hold on, z=cos(x); plot(x,z), hold off
```

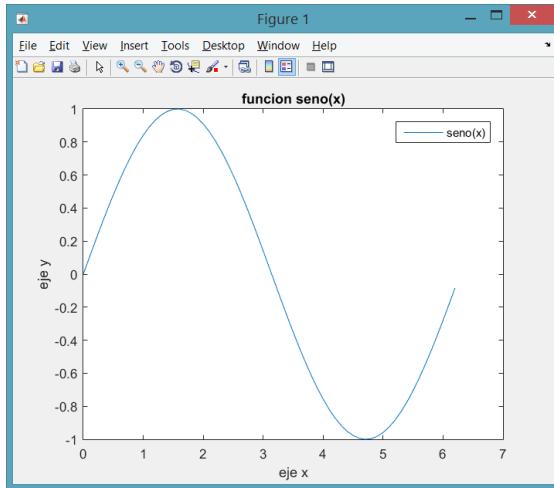
title



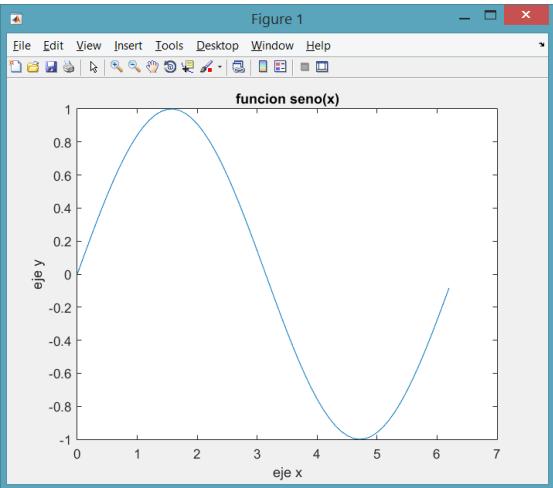
xlabel, ylabel

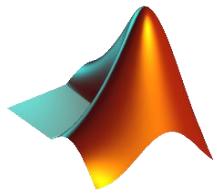


legend('leyenda1')

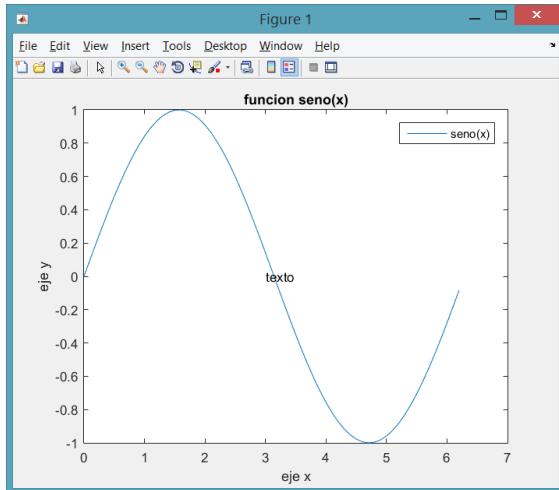


legend('off')

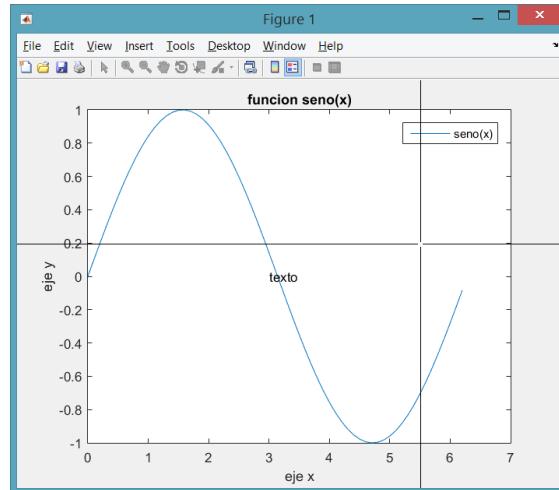




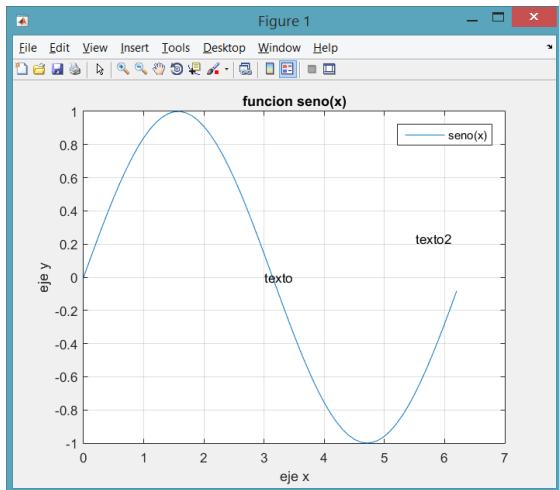
text



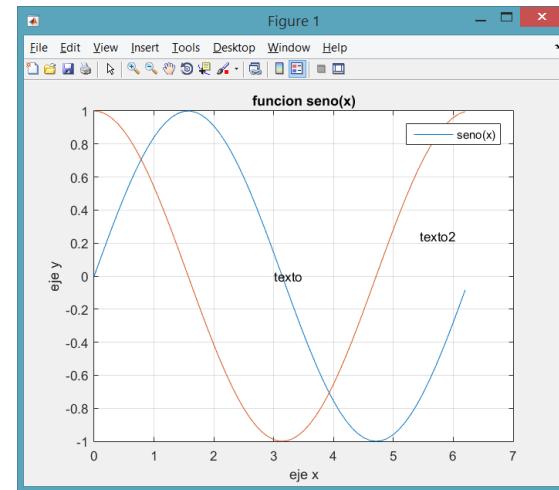
gtext

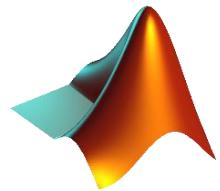


grid



hold on





Control de ventanas graficas

Si se llama a la función figure sin argumentos, se crea una ventana grafica con el número consecutivo que le corresponda. El valor de retorno es dicho número.

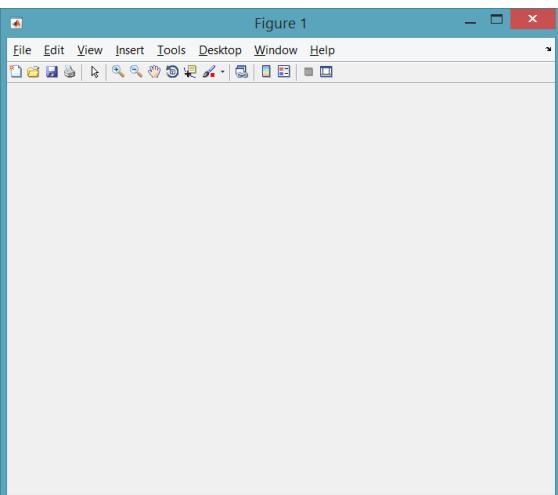
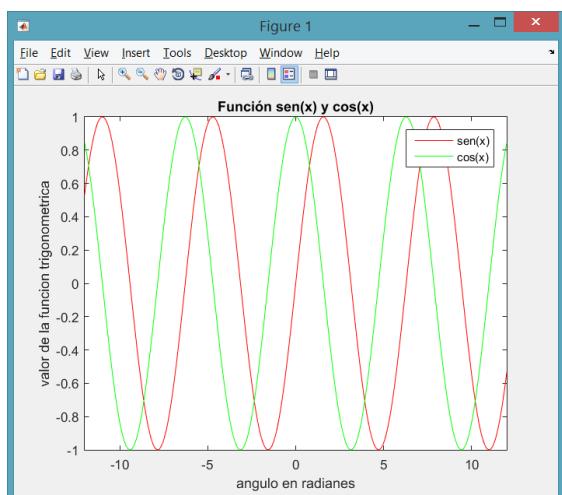
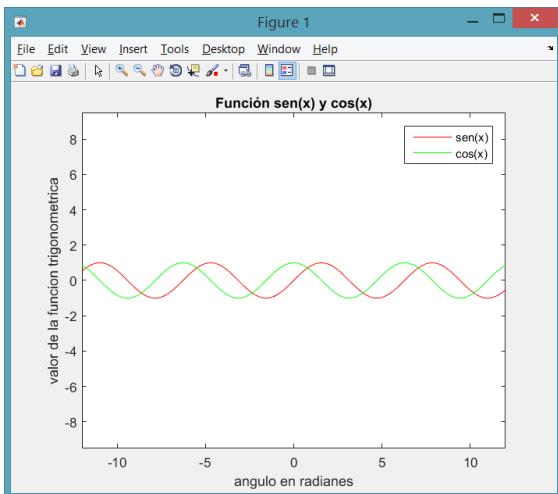
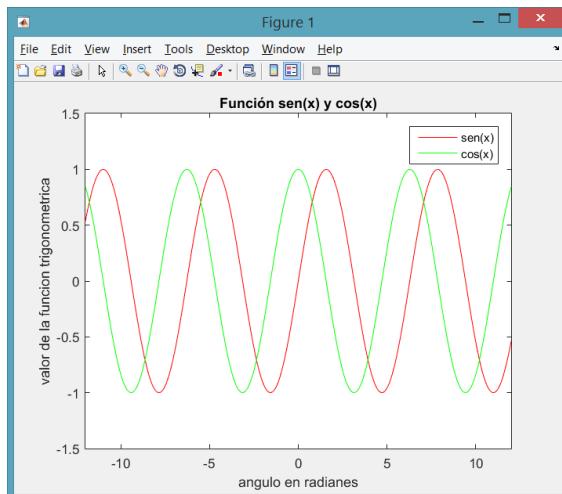
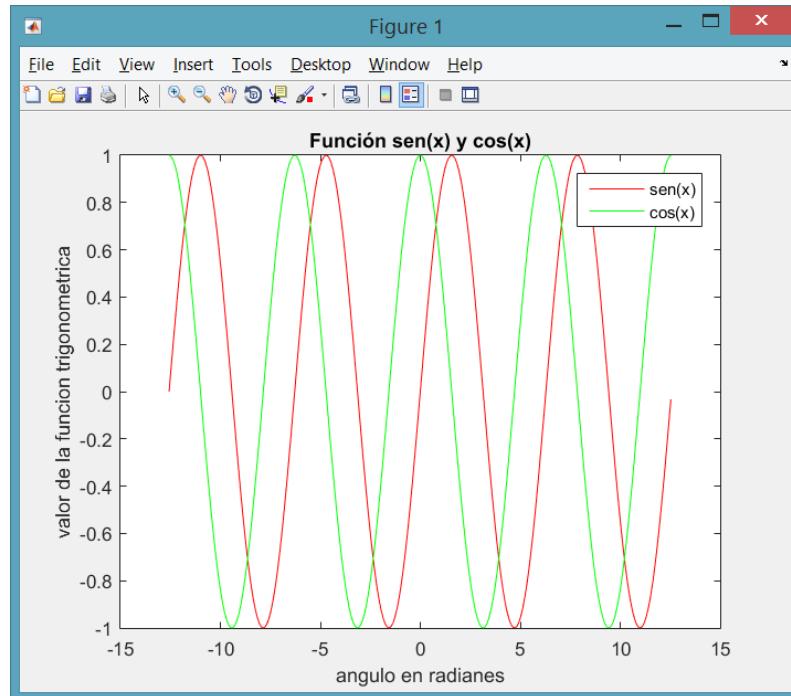
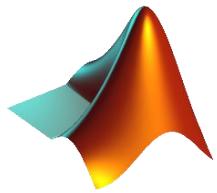
Por otra parte, el comando **figure(n)** hace que la ventana n pase a ser la ventana o figura activa. Si dicha ventana no existe, se crea una nueva ventana con el número indicado. La función **close** cierra la figura activa, mientras que **close (n)** cierra la ventana o figura numero n.

El comando **clf** elimina el contenido de la figura activa, es decir, la deja abierta pero vacía.

La función **gcf** devuelve el número de la figura activa en ese momento.

Ejecutemos las siguientes instrucciones

```
>> x=-4*pi:0.1:4*pi;
>> plot(x,sin(x), 'r', x,cos(x), 'g')
>> title('Función sen(x) y cos(x)')
>> legend('sen(x)', 'cos(x)')
>> xlabel('angulo en radianes'), figure(gcf)
>> ylabel('valor de la funcion trigonometrica'), figure(gcf)
>> axis([-12,12,-1.5,1.5]), figure(gcf)
>> axis('equal'), figure(gcf)
>> axis('normal'), figure(gcf)
>> clf
>> close
```



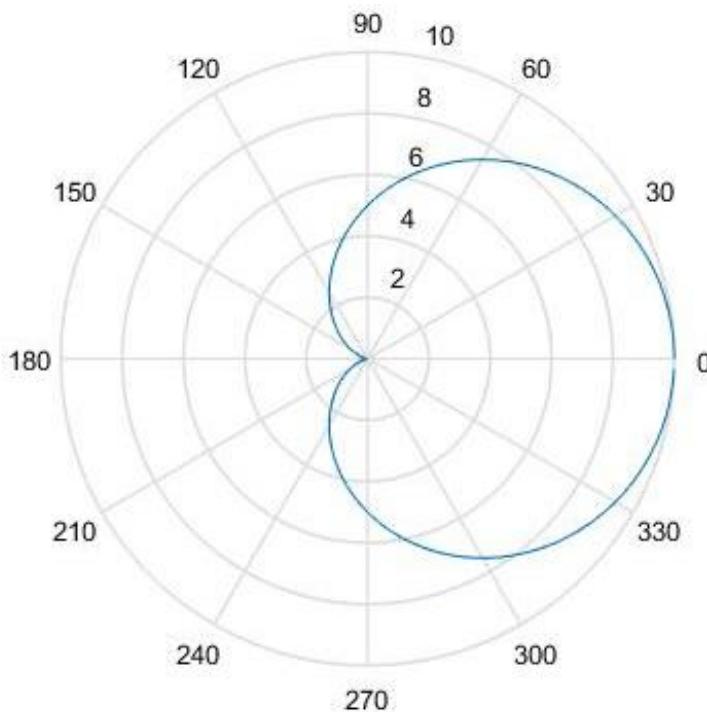
Gráficos en coordenadas polares

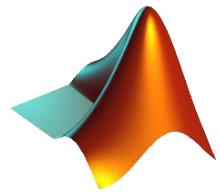
Matlab permite representar funciones en coordenadas polares. La sintaxis es,

polar(θ ,r) representa la curva en coordenadas polares $r=r(\theta)$
polar(θ ,r,S) representa la curva en coordenadas polares con el estilo de línea dado por **S**.

Ejemplo: Cardioide

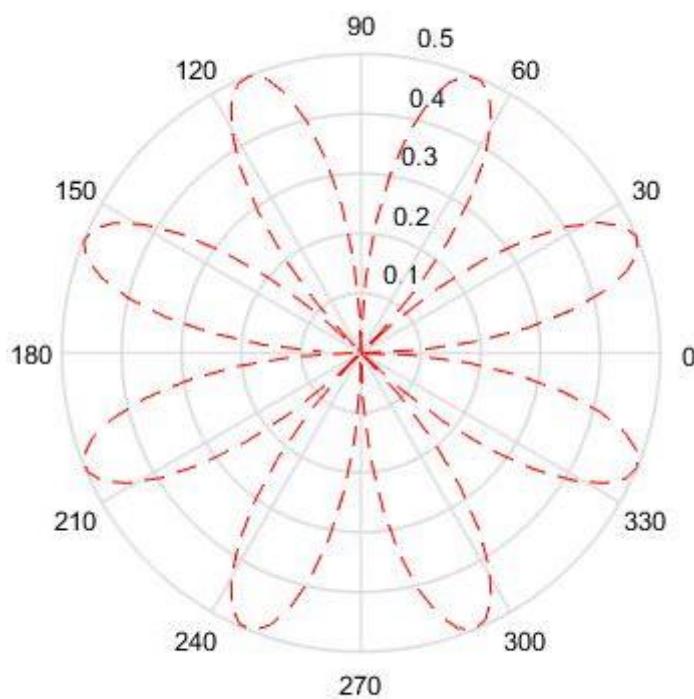
```
>> theta=0:1/20:2*pi;
>> r=5*(1+cos(theta));
>> polar(theta,r)
```





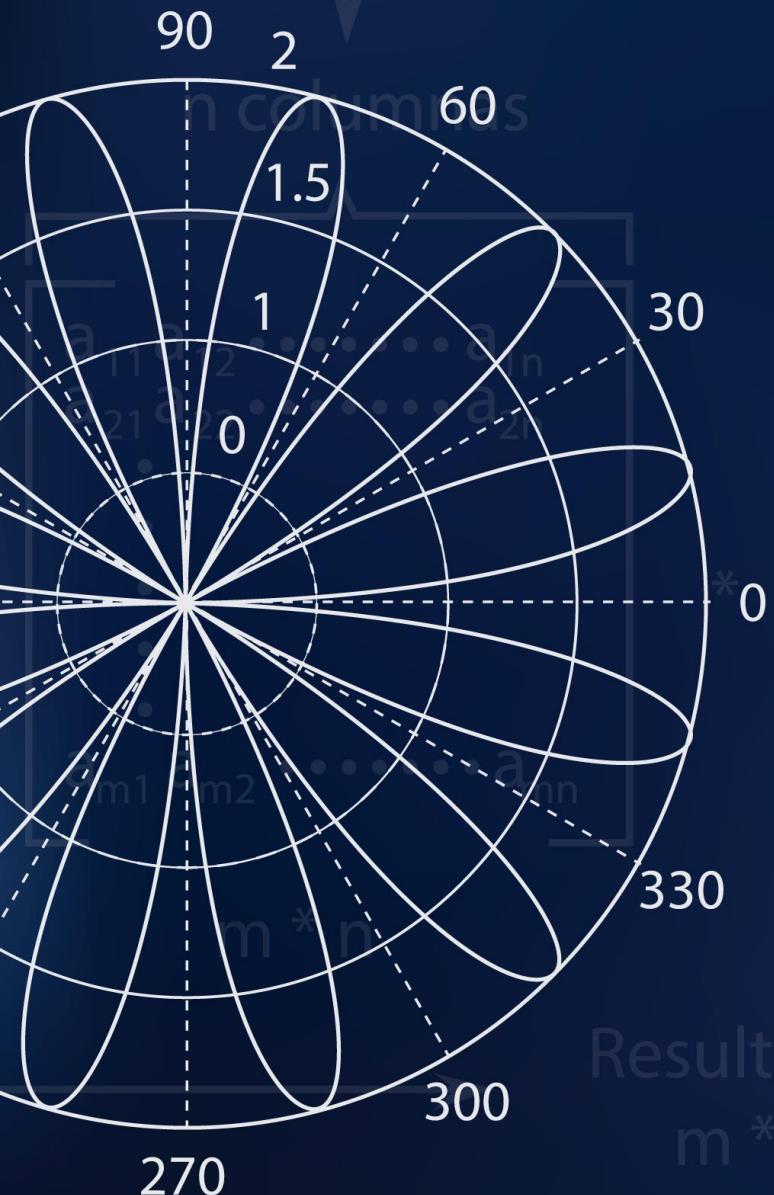
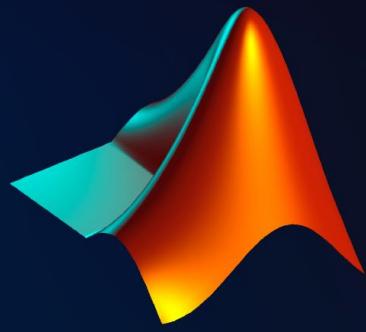
Ejemplo: Rosa de 8 pétalos

```
>> theta=0:0.05:2*pi;  
>> r=sin(2*theta).*cos(2*theta);  
>> polar(theta,r,'--r')
```

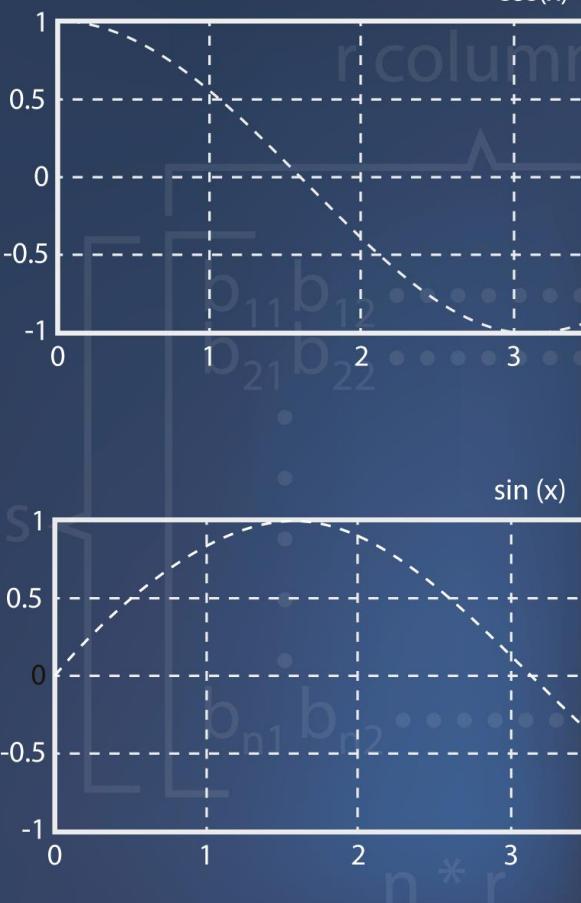


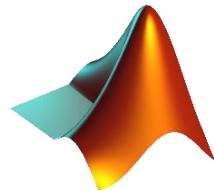
Curso MATLAB

Básico



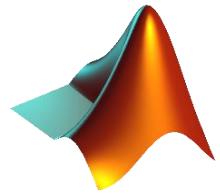
Resultado:
 $m * r$





Contenido

Variable simbólica	3
Límites	8
Derivadas	10
Integrales	11
Gráficas	13



Sesión 9: Cálculo simbólico

Variable simbólica

`syms de x y z ... t` convierte la variables x,y,z,...,t en simbólicas.

Ejemplos: convertir a forma simbólica las siguientes funciones y evaluarlas en algún punto.

$$g(x) = \sin(x) + x^2, \quad f(x) = \begin{pmatrix} x & x^3 \\ x^2 & x^4 \end{pmatrix}, \quad h(y, z) = y^2 + yz - 3$$

```
>> syms x
>> g(x)=sin(x)+x^2

g(x) =

sin(x) + x^2

>> g(pi/2)

ans =

pi^2/4 + 1

>> f(x)=[x x^3; x^2 x^4]

f(x) =

[ x, x^3]
[ x^2, x^4]

>> f(2)

ans =

[ 2,  8]
[ 4, 16]
```

```
>> syms y z
>> h(y,z)=y^2+y*z-3

h(y, z) =

y^2 + z*y - 3

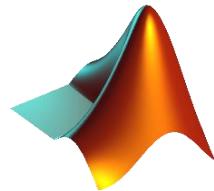
>> h(1,4)

ans =

2
```

Evalúa en los números indicados





syms lista las variables simbólicas en el espacio de trabajo.

```
>> syms
      'x'      'y'      'z'
```

x=sym('x') convierte la variable x en simbólica (equivale a **syms x**)

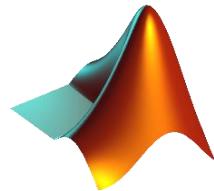
```
>> x=sym('x')
x =
x
>> f(x)=x+x^2
f(x) =
x^2 + x
>> f(2)
ans =
6
```

simplify simplifica las expresiones simbólicas declaradas.

Ejemplo: Simplificar la siguiente expresión

$$4ab + 2a^2 - ab$$

```
>> simplify(sym('4*a*b + 2*a^2 - a*b'))
ans =
a*(2*a + 3*b)
```



v=sym2poly(p) devuelve el vector numérico v con los coeficientes del polinomio simbólico p. El vector devuelto v incluye todos los coeficientes (incluido los iguales a cero).

Ejemplo: Obtener en un vector todos los coeficientes del polinomio

$$P(y) = y^4 - 5y^3 + y - 2$$

```
>> syms y
>> v = sym2poly(y^4 - 5*y^3 + y - 2)

v =
    1     -5      0      1     -2
```

P=polys2sym(v) crea un polinomio simbólico a partir de un vector de coeficientes v.

Ejemplo: Dados los coeficientes correspondientes del polinomio ordenado, obtener dicho polinomio de formas simbólica

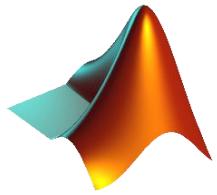
Coeficientes: 1, 2, 3, 4

```
>> syms x
>> p(x) = polys2sym([1, 2, 3, 4])

p(x) =
x^3 + 2*x^2 + 3*x + 4

>> p(1)

ans =
10
```



pretty(expr) convierte la expresión simbólica a escritura matemática.

Ejemplo: Dadas las siguientes funciones,

$$F = 2w + w^2 - 3$$

$$G = 3w^3 - 4w^2 + w$$

$$H = 5 - w^2 + 3w$$

Calcular,

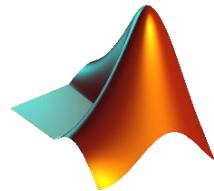
$$F+G+H, \quad F-H, \quad H/G$$

Usando la función pretty.

```
>> syms w
>> F=2*w + w^2 - 3;
>> G=3*w^3 - 4*w^2 + w;
>> H=5 - w^2 + 3*w;
>> pretty(F+G+H)
      3      2
3 w  - 4 w  + 6 w + 2

>> pretty(F-H)
      2
2 w  - w - 8

>> pretty(H/G)
      2
- w  + 3 w + 5
-----
      3      2
3 w  - 4 w  + w
```



expand(s) expande la expresión simbólica s.

Ejemplo: expandir la expresión, $(x - 2)(x + 4)$

```
>> syms x
>> expand( (x - 2)*(x + 4) )

ans =
x^2 - 6*x + 8
```

F=factor(x) devuelve todos los factores irreductibles de x en el vector F.

Ejemplo: Mostrar los factores irreductibles de la expresión

$$x^3 - 1$$

```
>> F=factor(x^3-1)

F =
[ x - 1, x^2 + x + 1]
```

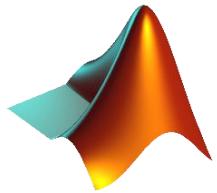
solve() resuelve ecuaciones algebraicas

Ejemplo: Resolver la siguiente ecuación simbólicamente

$$x^2 - 5 = 0$$

```
>> solve('x^2-5=0')

ans =
5^(1/2)
-5^(1/2)
```



Límites

A continuación conoceremos como Matlab nos permite calcular los límites de una sucesión y límites de funciones cuando van hacia un numero incluso por los laterales (límite por derecha e izquierda).

limit(sucesión, n, inf) calcula el límite de una sucesión, indicada por su término general, cuando n tiende al infinito.

Ejemplo: Calcular

$$\lim_{n \rightarrow \infty} \frac{3n^3 + 7n^2 + 1}{4n^3 - 8n + 5}$$

```
>> syms n
>> limit((3*n^3+7*n^2+1) / (4*n^3-8*n+5), n, inf)

ans =
3/4
```

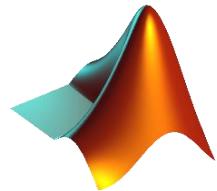
limit(función, x, a) calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende hacia el valor de a.

Ejemplo: Calcular

$$\lim_{x \rightarrow 1} \frac{-1 + x}{-1 + \sqrt{x}}$$

```
>> limit((-1+x) / (-1+x^(1/2)), x, 1)

ans =
2
```



limit(función,x,a,'right') calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende al valor de a por la derecha

Ejemplo: Calcular

$$\lim_{x \rightarrow 2^+} \frac{|x - 2|}{x - 2}$$

```
>> limit(abs(x-2) / (x-2), x, 2, 'right')

ans =
1
```

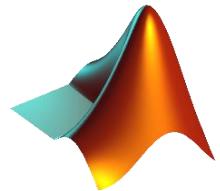
limit(función,x,a,'left') calcula el límite de la función de variable x, indicada por su expresión analítica, cuando la variable x tiende al valor de a por la izquierda.

Ejemplo: Calcular

$$\lim_{x \rightarrow 2^-} \frac{|x - 2|}{x - 2}$$

```
>> limit(abs(x-2) / (x-2), x, 2, 'left')

ans =
-1
```



Derivadas

syms x, diff(f,x) halla la función derivada de f respecto a x.

Ejemplo: Calcular,

$$\frac{d}{dx} \sin(x)$$

```
>> syms x, diff(sin(x),x)

ans =
cos(x)
```

Si nuestra función tuviera más de una variable, podemos proceder de la siguiente forma.

Ejemplo:

$$\frac{d}{dx} (x^3 + 2y^2 - 5x + 7y)$$

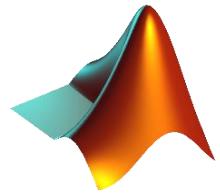
```
>> syms x y, diff(x^3+2*y^2-5*x+7*y,x)

ans =
3*x^2 - 5
```

syms x, diff(f,x,n) halla la derivada n-ésima de f respecto a x.

Ejemplo: Calcular

$$\frac{d^2}{dx^2} (x^3 + 2x^2 - 5x + 7)$$



```
>> syms x, diff(x^3+2*x^2-5*x+7,x,2)

ans =

6*x + 4
```

Veamos un ejemplo con 2 variables. Calcular,

$$\frac{d^2}{dx^2}(x^3 + 2x^2y^2 - 5x + 7y)$$

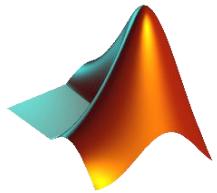
```
>> syms x y, diff(x^3+2*x^2*y^2-5*x+7*y,x,2)

ans =

4*y^2 + 6*x
```

Integrales

Comandos	Resultado
syms x, int(f(x),x)	Calcula la integral indefinida $\int f(x)dx$.
syms x y, int(int(f(x,y),x),y)	Calcula la integral doble $\iint f(x, y)dxdy$
syms x, int(f(x),x,a,b)	Calcula la integral definida $\int_a^b f(x)dx$
Syms xy a b c d, int(int(f(x,y),x,a,b),y,c,d)	Calcula la integral $\int_c^d \int_a^b f(x, y)dxdy$



Ejemplos: Calculemos las siguientes integrales simples

$$\int (x^2 + 3x)dx \quad \int_0^1 (x^2 + 3x)dx$$

```
>> syms x, int(x^2+3*x,x)

ans =

(x^2*(2*x + 9))/6

>> int(x^2+3*x,x,0,1)

ans =

11/6
```

Calcular las integrales dobles:

$$\iint xy^2 dx dy \quad \int_0^2 \int_0^1 xy^2 dx dy$$

```
>> syms x y
>> int(int(x*y^2,x),y)

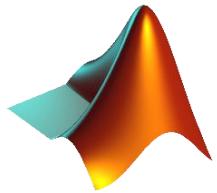
ans =

(x^2*y^3)/6

>> int(int(x*y^2,x, 0,1),y,0, 2)

ans =

4/3
```



Gráficas

En la sesión de gráficos bidimensionales presentamos al comando ezplot, en esta sesión lo usaremos para poder graficar funciones o resultados de las operaciones simbólicas que estemos trabajando.

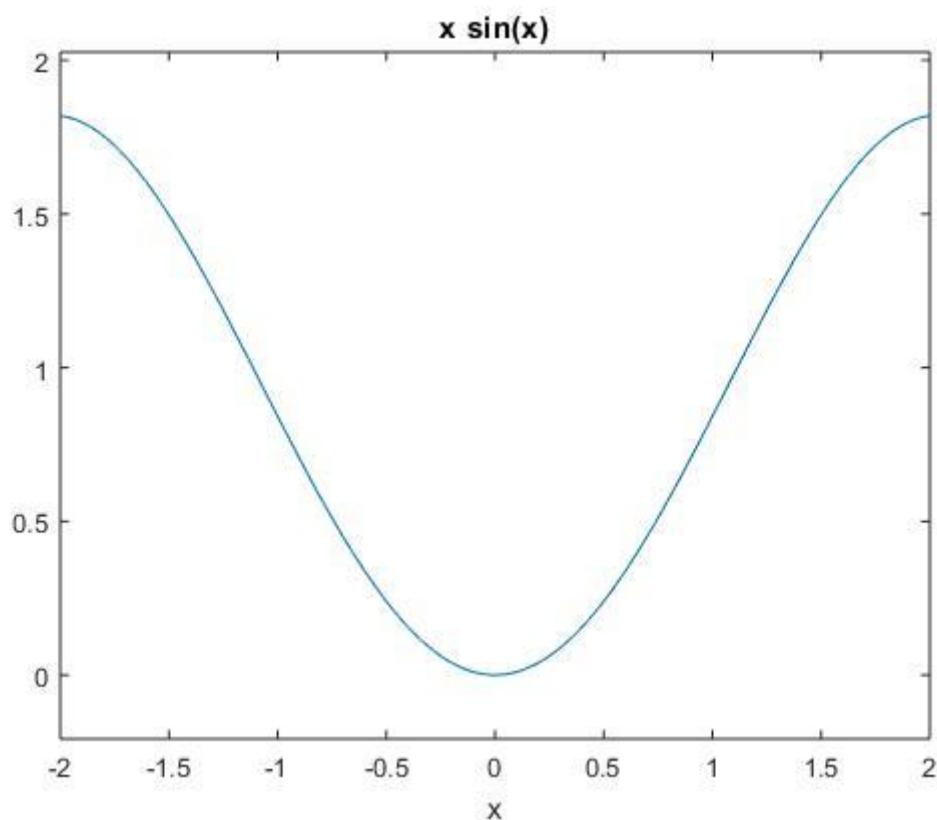
Veamos,

```
>> syms x
>> f=x*sin(x)

f =

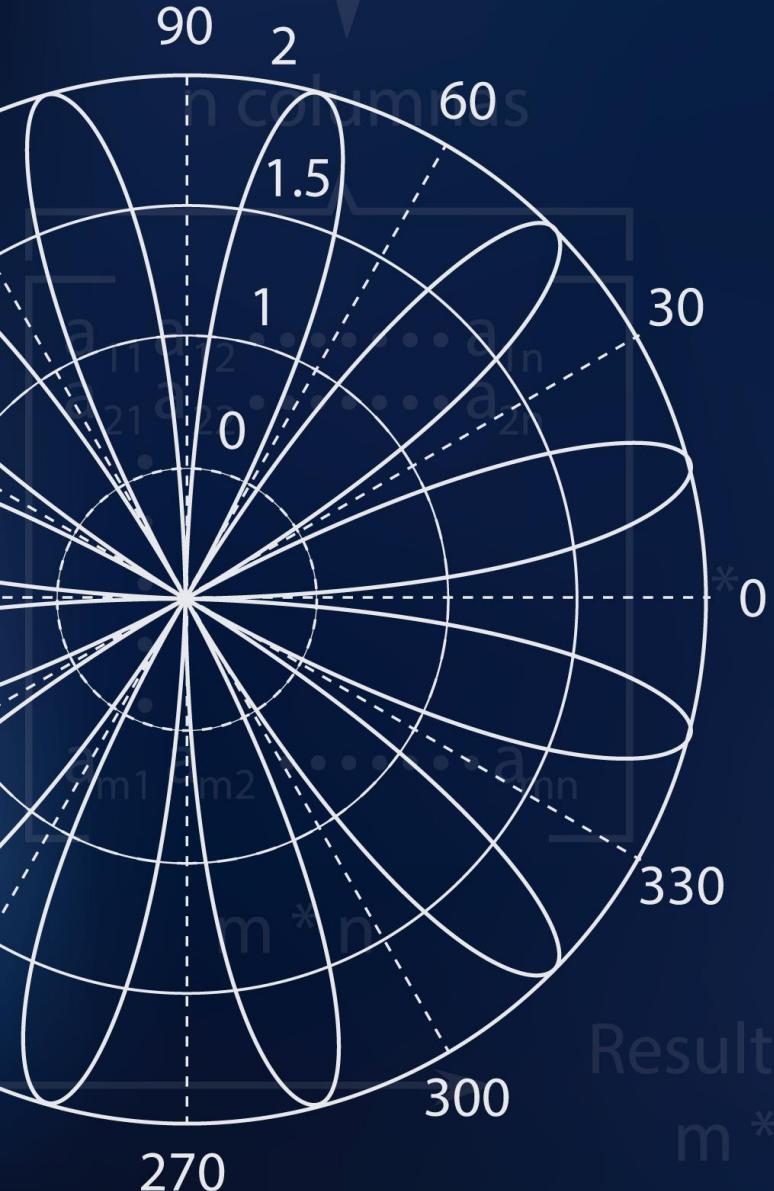
x*sin(x)

>> ezplot(f, [-2, 2])
```

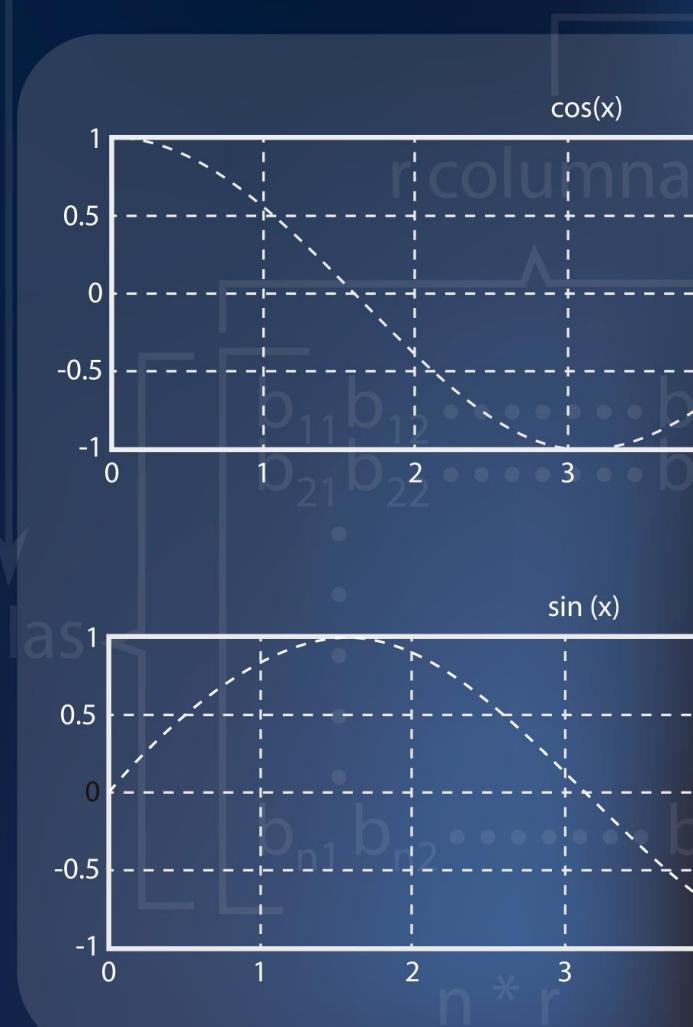
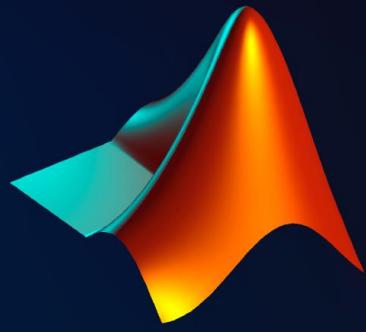


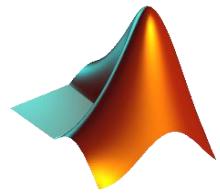
Curso MATLAB

Básico



Resultado:
 $m * r$





MATEMÁTICA COMPUTACIONAL Y APLICADA

MATLAB BÁSICO

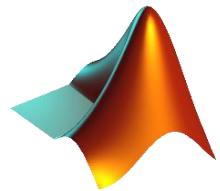
EVALUACIÓN FINAL

Nota: Resuelva cada una de las preguntas haciendo uso de los comandos aprendidos de MATLAB, salvo se indique el uso de algún otro comando.

1. Sea el polinomio: $x^4 + 1$
 - a. Hallar las raíces del polinomio.
 - b. Usar los comandos aprendidos y expresar las raíces en su forma trigonométrica.
 - c. Derivar el polinomio, y expresar el resultado a escritura matemática.
2. $x(t) = t^2 + 2t + 1$, representa la posición de un móvil respecto del tiempo, x en metros y t en segundos.
 - a. Hallar la función de velocidad y calcular la velocidad del móvil en el tiempo $t = 10s$.
 - b. Graficar en diferentes subventanas: posición vs tiempo y velocidad vs tiempo, indicando título, ejes y leyenda.
3. Sea el sistema de 3 ecuaciones lineales:

$$\begin{array}{rcl} 2x & + & 3y & + & z & = & 3 \\ x & + & 2y & + & z & = & 1 \\ -x & + & 4y & + & 0 & = & -2 \end{array}$$

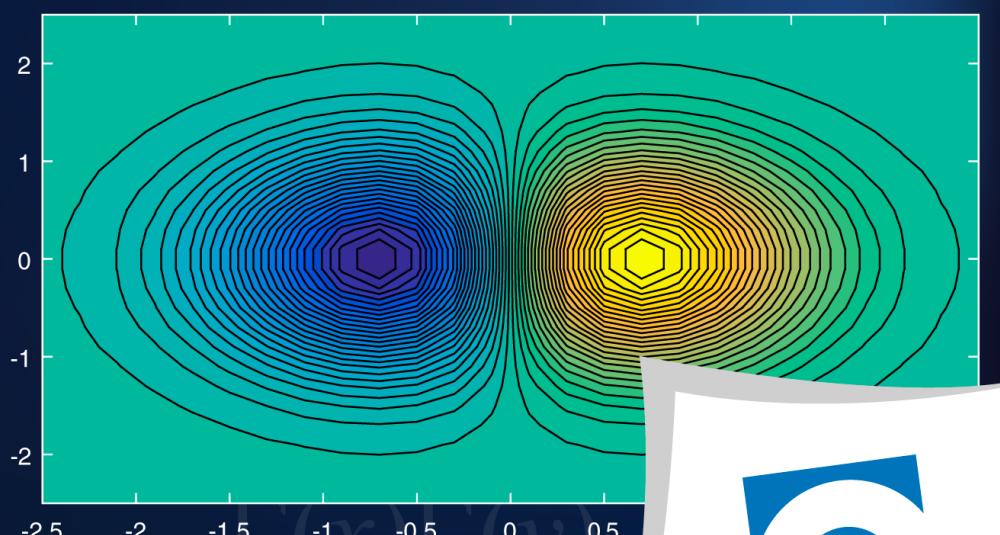
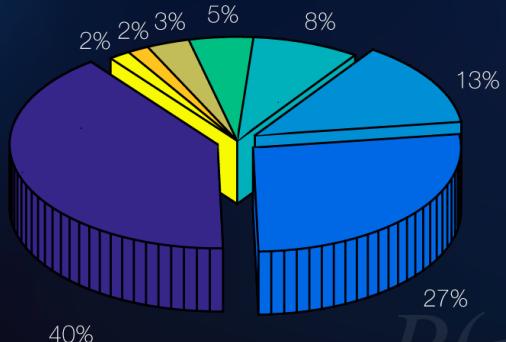
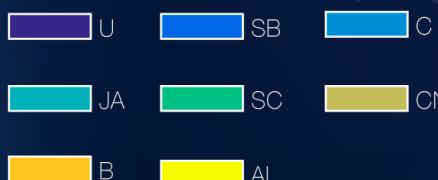
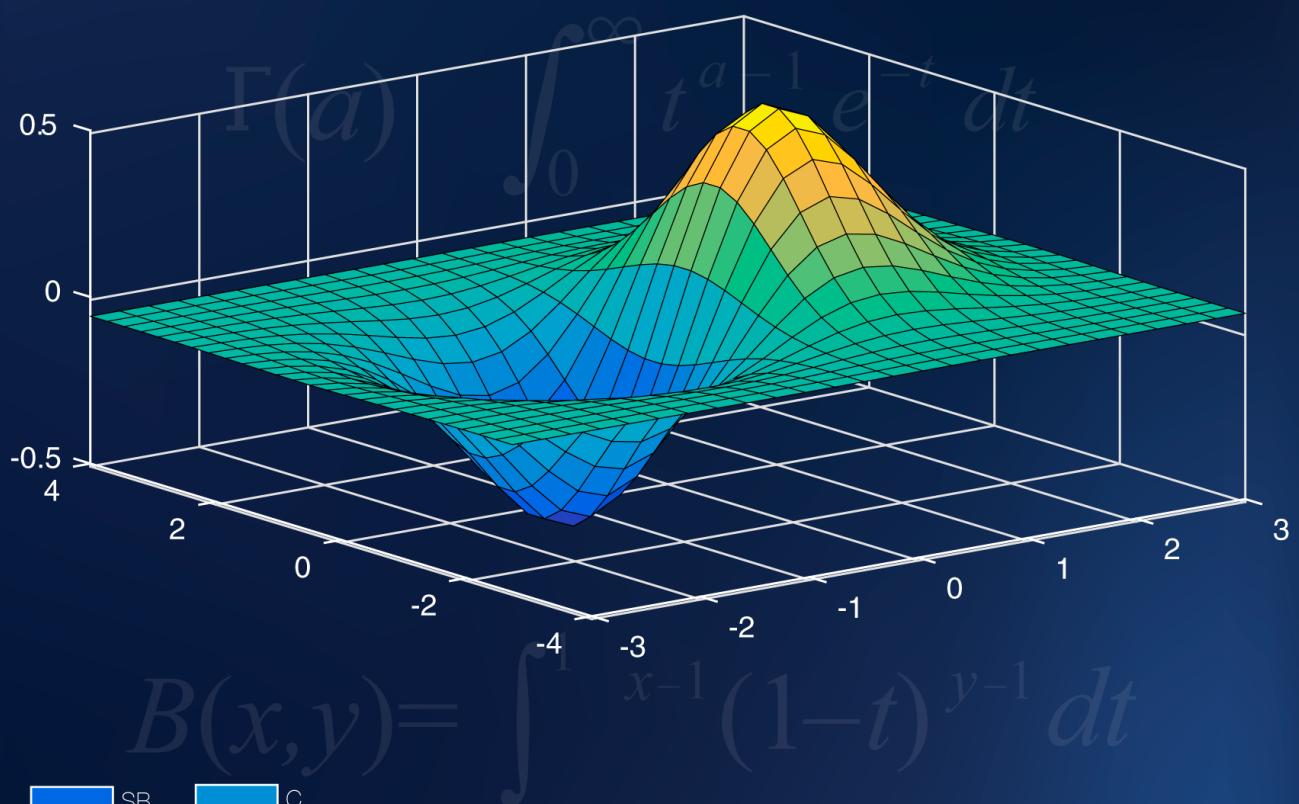
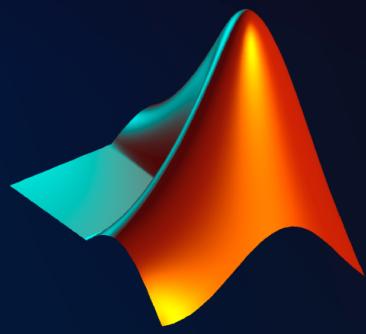
- a. Verifique que el sistema tiene solución y realizando operaciones fila hallar la solución, luego comprobar el resultado usando el comando correspondiente de MATLAB.
- b. Resolver si la matriz de coeficientes es invertible, de ser así hallar su inversa realizando operaciones fila y comprobar con el comando correspondiente de MATLAB.

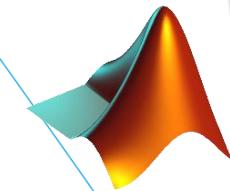


- c. Calcular los *autovalores* y *autovectores* correspondientes de la matriz de coeficientes.
4. Generar dos vectores con elementos enteros de manera aleatoria de dimensiones 4 y 3, formar un vector v usando los vectores obtenidos, luego ordenarlo en forma descendente sin hacer uso del comando *sort*.

Curso MATLAB

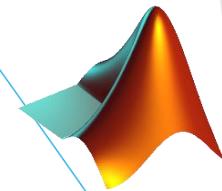
Intermedio





Contenido

Funciones matemáticas especiales	3
Funciones de bessel.....	3
Funciones de gamma.....	6
Funciones de beta	7
Funciones de legendre	7
Funciones para conversión de sistemas de coordenadas	11
Cartesiana a cilíndricas, polares y esféricas	13
Cilíndricas a cartesianas, polares a cartesianas y esféricas a cartesianas	13



Sesión 1: Funciones especiales y sistemas de coordenadas

Funciones matemáticas especiales

Funciones de bessel

La ecuación diferencial de segundo orden

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} + (x^2 - v^2) y = 0$$

Se conoce como la ecuación de bessel. La solución de esta ecuación diferencial se escribe

$$y = AJ_v(x) + BY_v(x)$$

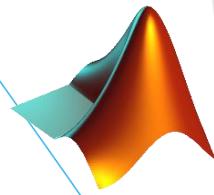
Donde A y B son constantes que determinan a partir de las condiciones iniciales. El índice v es un número real

$$J_v(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{v+2k}}{k!(v+k)!}, \quad Y_v(x) = \frac{J_v(x)\cos(vx) - J_{-v}(x)\cos(vx)}{\sin(vx)}$$

Para v entero se cumple

$$J_{-v}(x) = (-1)^v J_v(x)$$

$$Y_{-v}(x) = (-1)^v Y_v(x)$$

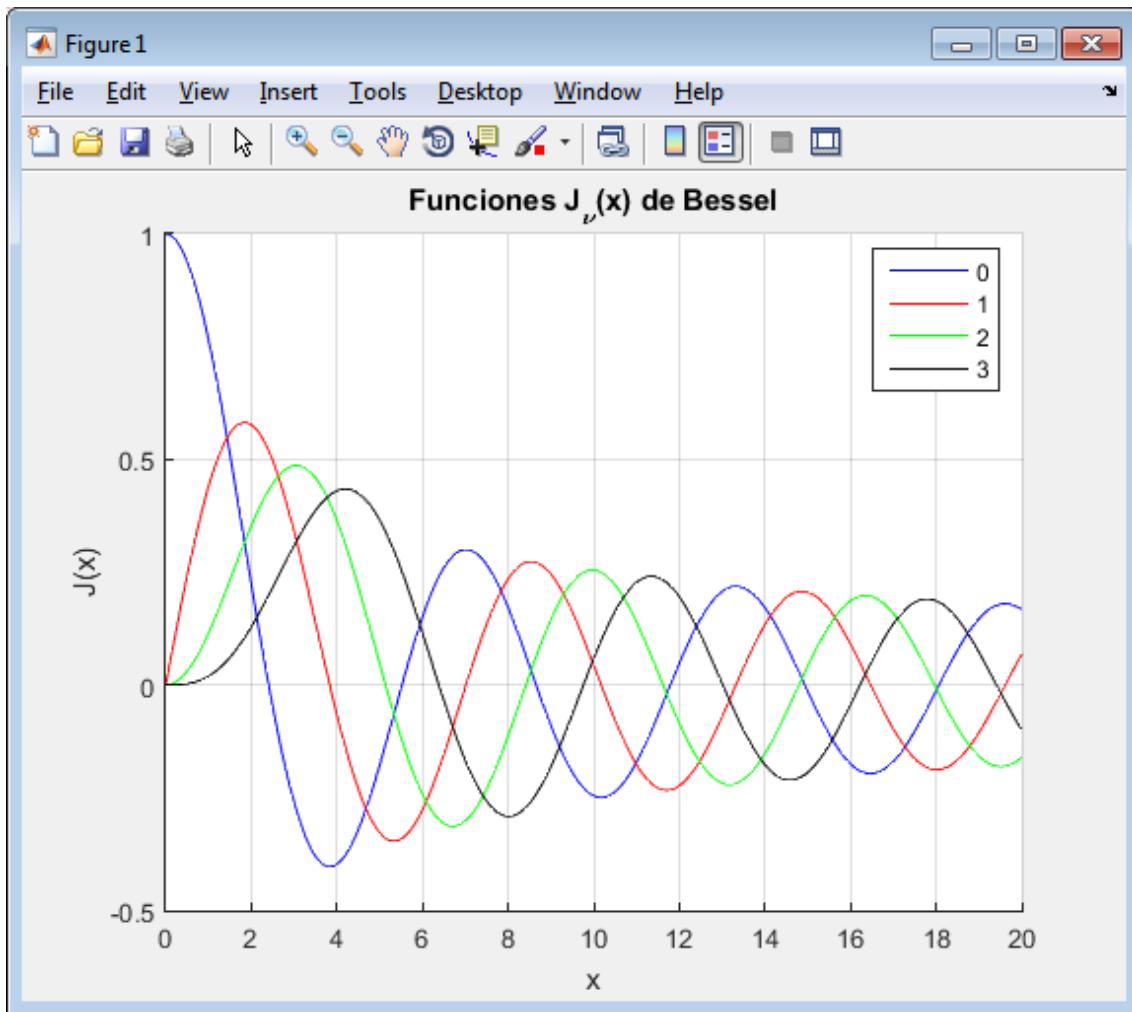


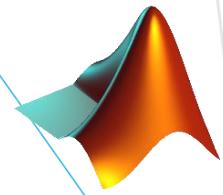
besselj(k,Z)	Funciones de bessel de 1 ^a y 2 ^a clase soluciones de la ecuación de bessel
bessely(k,Z)	$z^2w''+zw'+(z^2-k^2)w=0, w=w(z)$

besseli(k,Z)	Funciones de bessel de 1 ^a y 2 ^a clase modificadas soluciones de la ecuación de bessel
besselk(k,Z)	$z^2w''+zw'+(z^2+k^2)w=0, w=w(z)$

Funciones de Bessel de primera especie

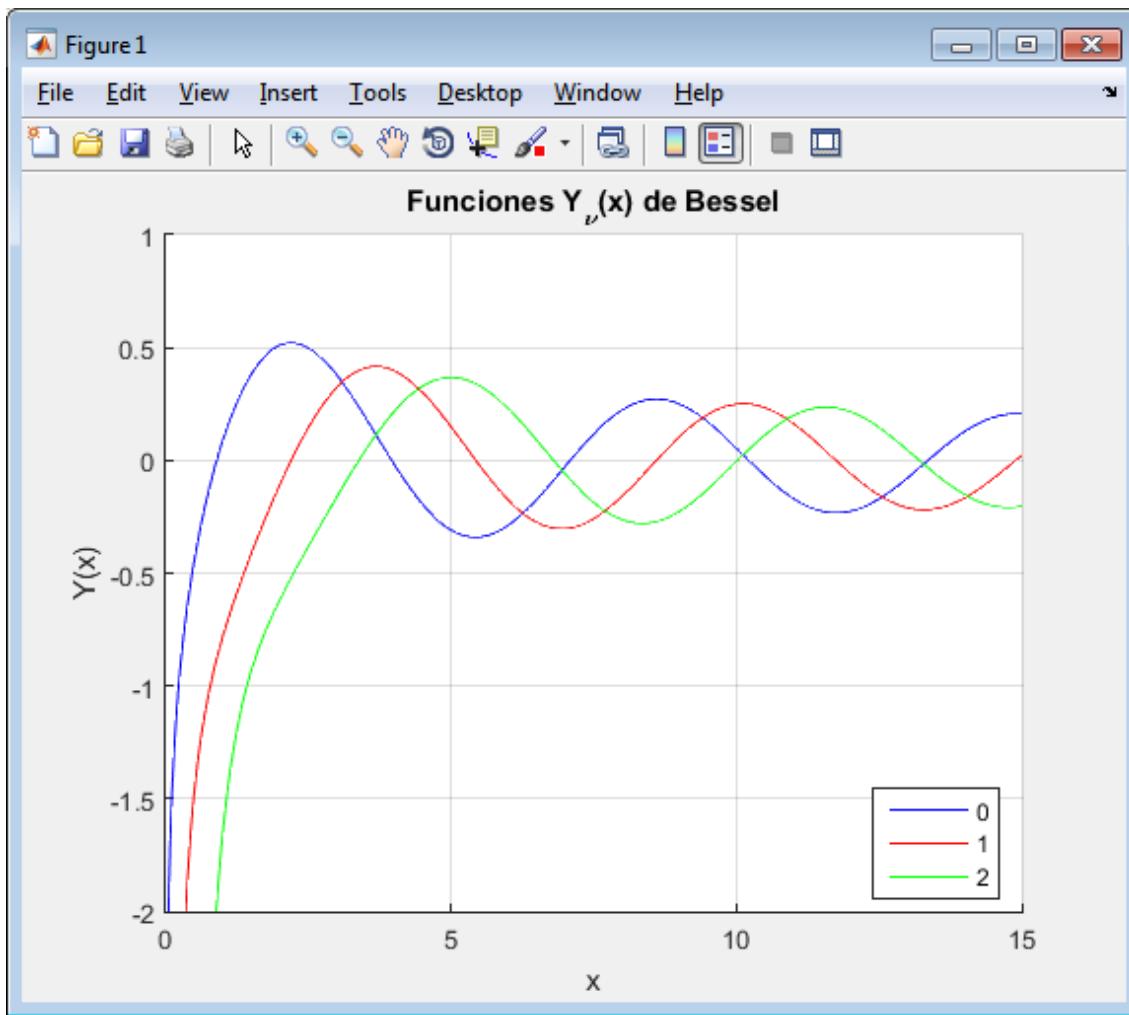
Graficamos $J_0(x)$, $J_1(x)$, $J_2(x)$ y $J_3(x)$ con el uso de la función **besselj(n,x)** de MATLAB, dando valores a su primer parámetro **n=0,1,2,3**.

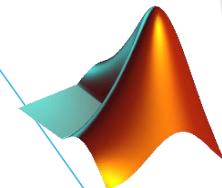




Funciones de Bessel de segunda especie

Representamos $Y_0(x)$, $Y_1(x)$, $Y_2(x)$ llamando a la función **bessely(n,x)** de MATLAB, dando valores a su primer parámetro **n=0,1,2**.





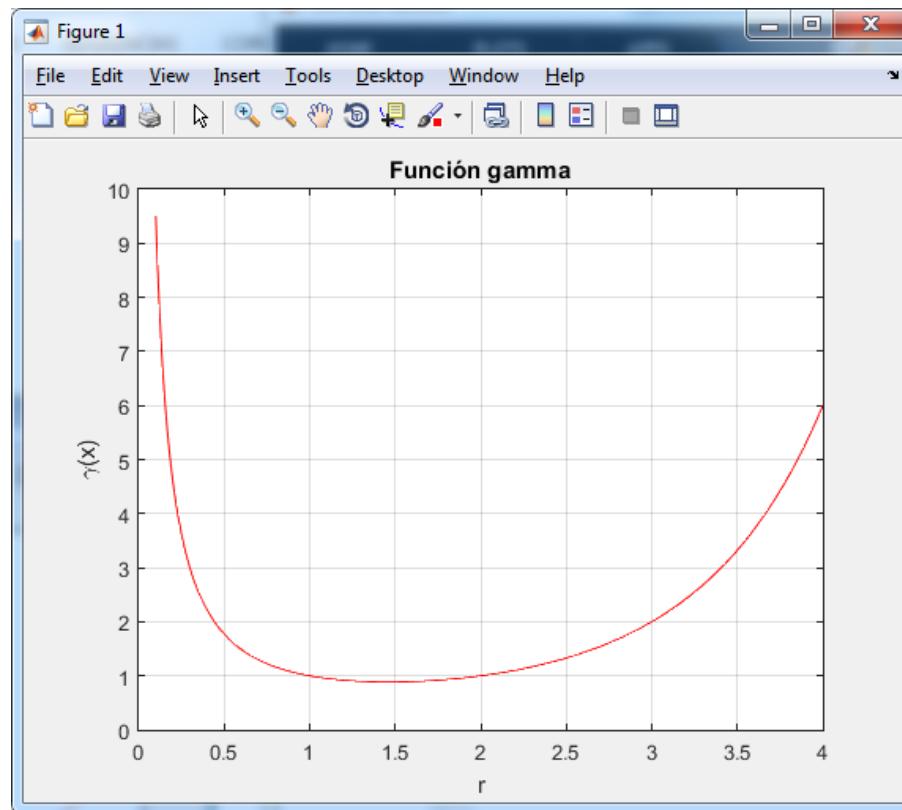
Funciones de gamma

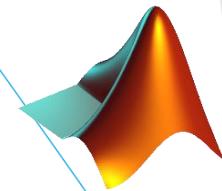
Se define del siguiente modo $\Gamma(z) = \int_0^{\infty} e^{-t} t^{z-1} dt$

Donde **z** puede ser un número real o complejo. Veamos el caso en el que **z** sea real y positivo.

Representamos gráficamente la función gamma para **z>0**, llamando a la función del mismo nombre.

```
>> x=0.1:0.01:4;
>> y=gamma(x);
>> plot(x,y,'r')
>> grid
>> xlabel('r')
>> ylabel('\gamma(x)')
>> title('Función gamma')
fx >>
```





Funciones de beta

La función beta está estrechamente relacionado a la función gamma.

beta(Z,W)	Función beta
	$\beta(z, w) = \int_0^1 (1-t)^{w-1} t^{z-1} dt = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)}$
betainc(X,Z,W)	Beta incompleta
	$I_x(z, w) = \frac{1}{\beta(z, w)} \int_0^x (1-t)^{w-1} t^{z-1} dt$
betaIn(X,Z,W)	Logaritmo de la función beta $\log \beta(z, w)$

Funciones de legendre

Los primeros polinomios de Legendre son.

$$P_0(x) = 1$$

$$P_1(x) = x$$

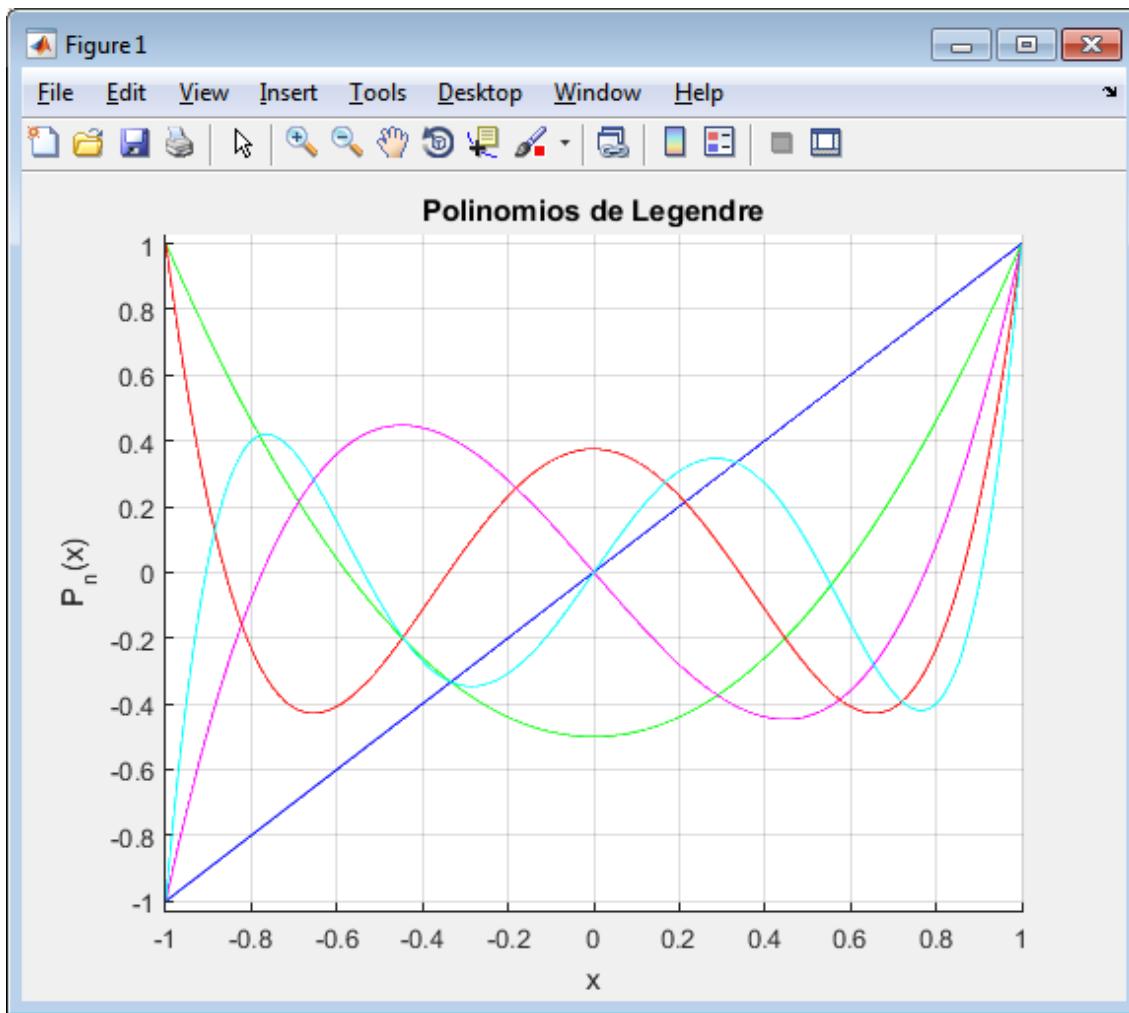
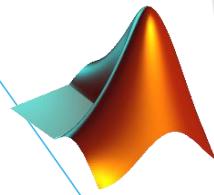
$$P_2(x) = \frac{1}{2}(3x^2 - 1)$$

$$P_3(x) = \frac{1}{2}(5x^3 - 3x)$$

$$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3)$$

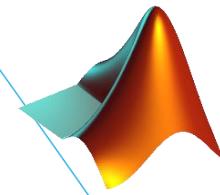
$$P_5(x) = \frac{1}{8}(63x^5 - 70x^3 + 15x)$$

.....



P=legendre(n,X)

Funciones asociadas a legendre



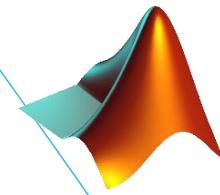
Otros

pow2(X)	<p>Si X es un número, se obtiene 2 elevado a ese número.</p> <p>Si X es un vector (o matriz), se obtiene un vector (o matriz) con elementos que son iguales a 2 elevado al elemento con la misma posición del vector (o matriz).</p> 2^y
----------------	--

```
Command Window
>> pow2 (3)
ans =
8

>> pow2 ([1 2 3])
ans =
2     4     8

>> pow2 ([1 2 3; 4 5 6])
ans =
2     4     8
16    32    64
fx >>
```

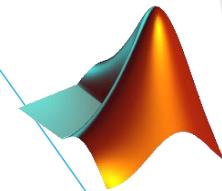


rats(X)

Si **X** es un número, vector o matriz; **rats(x)** devuelve en formato racional cada elemento de **X**.

Command Window

```
>> x=[0.5 0.36 2.84]  
  
x =  
  
    0.5000    0.3600    2.8400  
  
>> rats (x)  
  
ans =  
  
    1/2          9/25        71/25  
  
>> y=[1.4 0.6; -3.04 8.123]  
  
y =  
  
    1.4000    0.6000  
   -3.0400    8.1230  
  
>> rats (y)  
  
ans =  
  
    7/5          3/5  
   -76/25      1519/187  
  
fx >>
```



Funciones para conversión de sistemas de coordenadas

Matlab tiene implementadas funciones para hacer conversiones de coordenadas de unos sistemas a otros permitiéndonos trabajar con sistema de coordenadas. A continuación haremos una breve aclaración de los sistemas de coordenadas menos habituales que son las cilíndricas y las esféricas.

En un sistema de **coordenadas cilíndricas**, un punto \mathbf{P} del espacio es representado por una tripleta ordenada (r, θ, z) , donde:

r : es la distancia del origen a la proyección de \mathbf{P} (\mathbf{P}') en el plano XY.

θ : es el ángulo entre el eje X y el segmento \mathbf{OP}' .

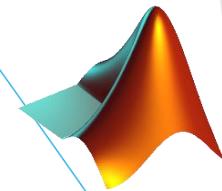
z : es la distancia \mathbf{PP}' .

En un sistema de **coordenadas esféricas**, un punto \mathbf{P} del espacio es representado por una tripleta ordenada (ρ, θ, ϕ) , donde:

ρ : es la distancia de \mathbf{P} al origen.

θ : es el mismo ángulo que el usado en coordenadas cilíndricas.

ϕ : es el ángulo entre el eje Z positivo y el segmento \mathbf{OP} .



Fácilmente se llega a las ecuaciones de conversión siguientes:

Cilíndricas a rectangulares:

$$\begin{cases} x = r \cos(\theta) \\ y = r \sin(\theta) \\ z = z \end{cases}$$

Rectangulares a cilíndricas:

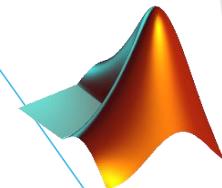
$$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \operatorname{arctg} \frac{y}{x} \\ z = z \end{cases}$$

Esféricas a rectangulares:

$$\begin{cases} x = \rho \sin\phi \cos\theta \\ y = \rho \sin\phi \sin\theta \\ z = \rho \cos\phi \end{cases}$$

Rectangulares a esféricas:

$$\begin{cases} \rho = \sqrt{x^2 + y^2 + z^2} \\ \theta = \operatorname{arctg} \frac{y}{x} \\ \phi = \operatorname{arcsen} \frac{z}{\rho} \end{cases}$$



Cartesiana a cilíndricas, polares y esféricas

Matlab nos permite convertir directamente coordenadas cartesianas a cilíndricas, cartesianas a polares o, cartesianas a esféricas.

<code>[theta,rho,z]=cart2pol(x,y,z)</code>	Transforma cartesianas a cilíndricas.
<code>[theta,rho]=cart2pol(x,y)</code>	Transforma cartesianas a polares.
<code>[theta,phi,r]=cart2sph(x,y,z)</code>	Transforma cartesianas a esféricas.

Cilíndricas a cartesianas, polares a cartesianas y esféricas a cartesianas

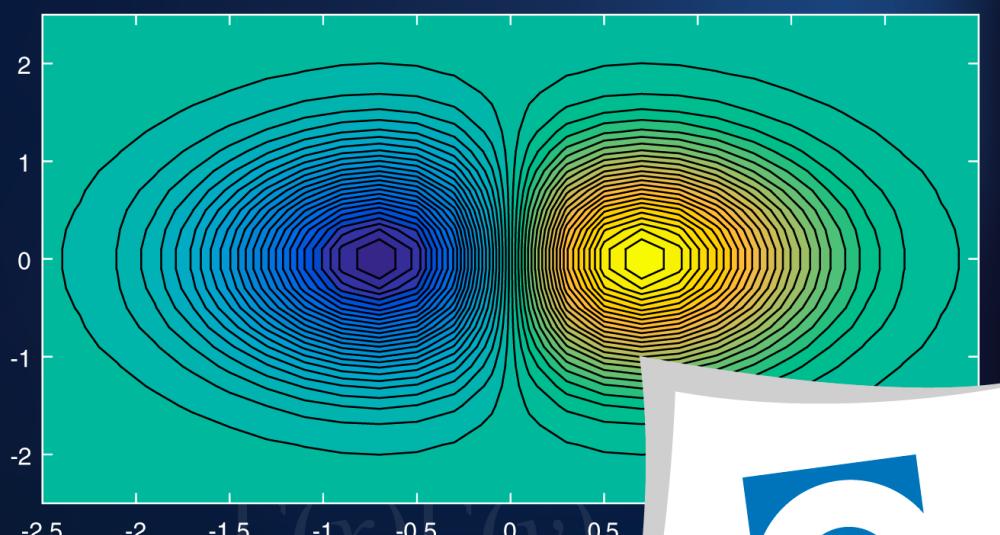
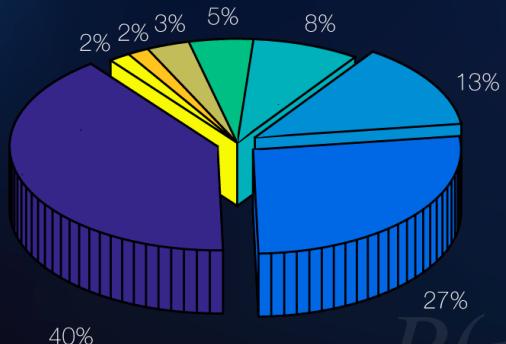
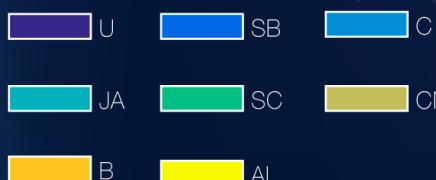
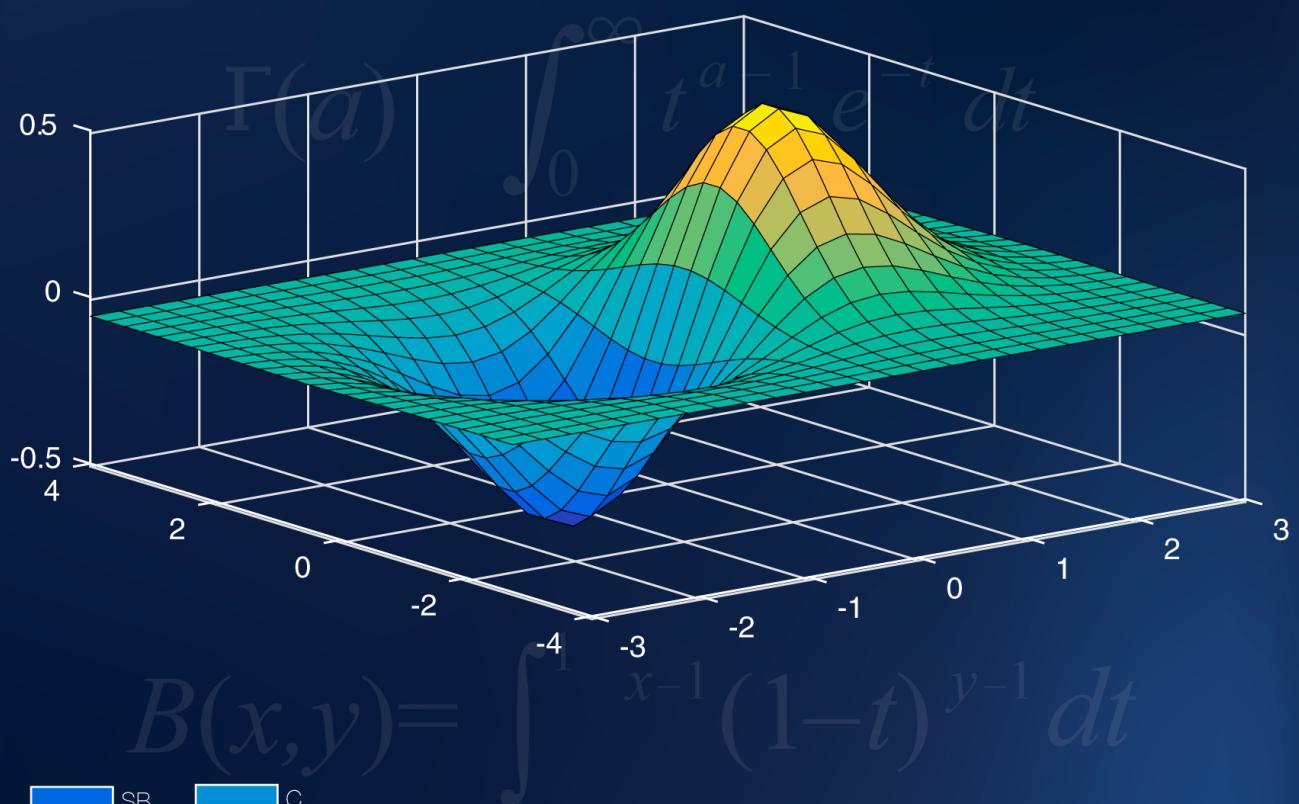
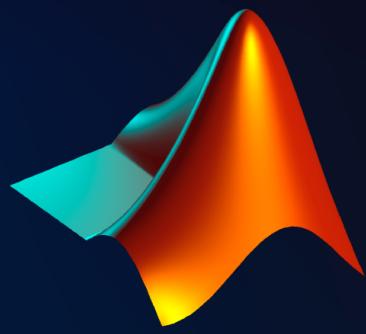
Matlab también nos permite convertir directamente de coordenadas cilíndricas a cartesianas, polares a cartesianas y esféricas a cartesianas.

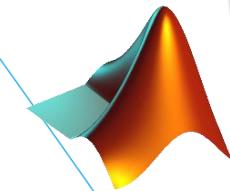
<code>[x,y,z]=pol2cart(theta,rho,z)</code>	Transforma cilíndricas a cartesianas.
<code>[x,y]=pol2cart(theta,rho)</code>	Transforma polares a cartesianas.
<code>[x,y,z]=sph2cart(theta,phi,r)</code>	Transforma esféricas a cartesianas.

Si nuestro interés es convertir por ejemplo de coordenadas polares a esféricas, o cilíndricas a polares, debemos hacer uso de las combinaciones de los comandos mostrados en las tablas. Con esto es posible convertir coordenadas de un sistema a otro.

Curso MATLAB

Intermedio





Contenido

Funciones estadísticas.....	3
Máximos, mínimos, medias y medianas	3
Desviaciones, cuasidesviaciones, varianzas y cuasivarianzas	6
Gráficos estadísticos.....	8
Diagrama de Pareto	8
Diagrama de barras.....	9
Histogramas	12
Graficas de sectores	14

Sesión 2: Funciones estadísticas

Funciones estadísticas

Máximos, mínimos, medias y medianas

max(a), min(a), median(a), si **a** es un vector devuelve el valor máximo, mínimo, o de la mediana respectivamente. Si **a** es una matriz devuelve el valor máximo, mínimo o de la mediana respectivamente de cada columna.

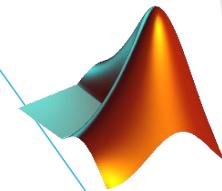
Ahora veamos algunos tipos de medias.

mean(a), si **a** es un vector (o matriz) devuelve el valor de la media aritmética de los elementos (o cada columna).

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

geomean(a), si **a** es un vector (o matriz) devuelve el valor de la media geométrica de los elementos (o cada columna).

$$\bar{x} = \left(\prod_{i=1}^n x_i \right)^{1/n}$$



harmmean(a), si **a** es un vector (o matriz) devuelve el valor de la media harmónica de los elementos (o cada columna). Es el inverso de la media aritmética de los recíprocos de una cantidad finita de números.

$$\bar{x} = n \times \left(\sum_{i=1}^n \frac{1}{x_i} \right)^{-1}$$

Ejemplo: **a** es un vector.

```
>> a=[5 2 12 6 15 7 11];
>> max=max(a), min=min(a), mediana=median(a), ...
m_a=mean(a), m_g=geomean(a), m_h=harmmean(a)

max =
15

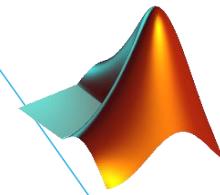
min =
2

mediana =
7

m_a =
8.2857

m_g =
7.0097

m_h =
5.5981
```



Ejemplo: a es una matriz

```
>> clear
>> a=[3 7 2; 5 3 8; 2 11 4],...
max=max(a), min=min(a), mediana=median(a),...
m_a=mean(a), m_g=geomean(a), m_h=harmmean(a)

a =
3      7      2
5      3      8
2      11      4

max =
5      11      8

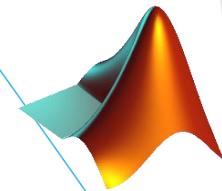
min =
2      3      2

mediana =
3      7      4

m_a =
3.3333    7.0000    4.6667

m_g =
3.1072    6.1358    4.0000

m_h =
2.9032    5.2901    3.4286
```



Desviaciones, cuasidesviaciones, varianzas y cuasivarianzas

Desviación típica

Std(a,1), Si **a** es un vector (o matriz) devuelve el valor de la desviación típica de los elementos (o cada columna).

$$S_N = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$$

Cuasidesviación típica

Std(a), Si **a** es un vector (o matriz) devuelve el valor de la cuasidesviación típica de los elementos (o cada columna).

$$S_{N-1} = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2}$$

Varianza

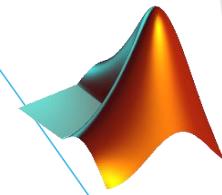
var(a,1), si **a** es un vector, retorna la varianza de los valores. Si **a** es una matriz retorna la varianza de cada columna.

$$S_N^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2$$

Cuasivarianza

var(a), si **a** es un vector, retorna la cuasivarianza corregida de los valores. Si **a** es una matriz retorna la cuasivarianza de cada columna.

$$S_{N-1}^2 = \frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2$$



Ejemplo: a es una matriz

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

a =

    9     8     7
    6     5     4
    3     2     1

>> std(a,1)

ans =

    2.4495    2.4495    2.4495

>> std(a)

ans =

    3     3     3

>> var(a,1)

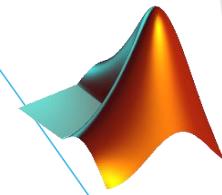
ans =

    6     6     6

>> var(a)

ans =

    9     9     9
```

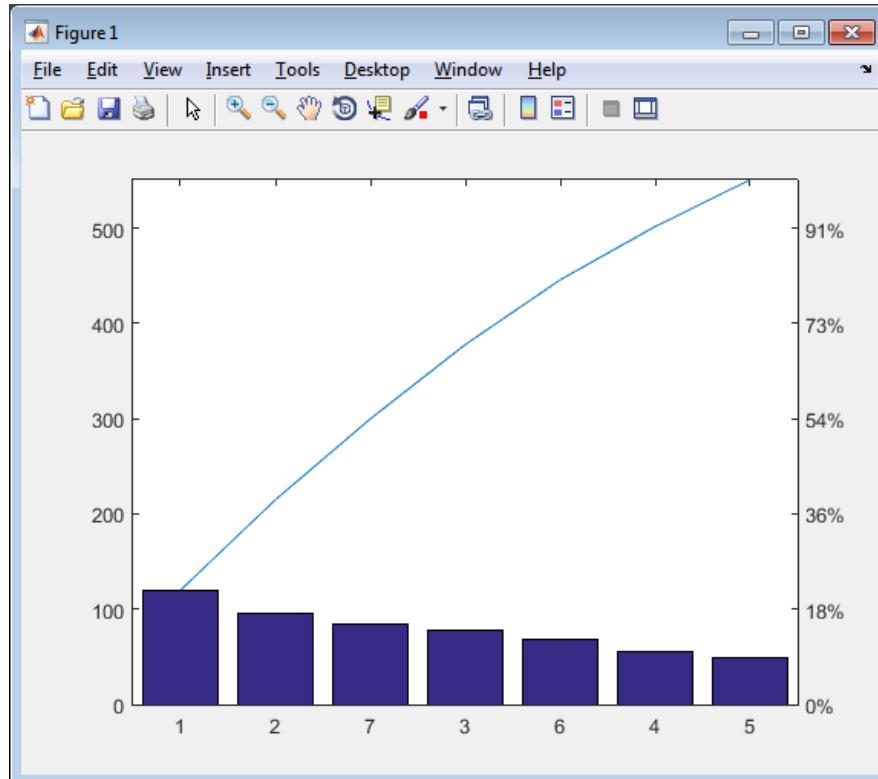


Gráficos estadísticos

Diagrama de Pareto

El diagrama de Pareto que produce Matlab consta de barras cuyas alturas son el número de escaños, ordenados en forma decreciente y sobre las barras, un polígono con las frecuencias acumuladas de los escaños. Además, en el eje vertical derecho aparece una escala de porcentajes.

```
>> x=[120 95 78 56 49 68 85 ]  
  
x =  
  
120      95      78      56      49      68      85  
  
>> pareto(x)
```



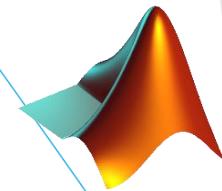


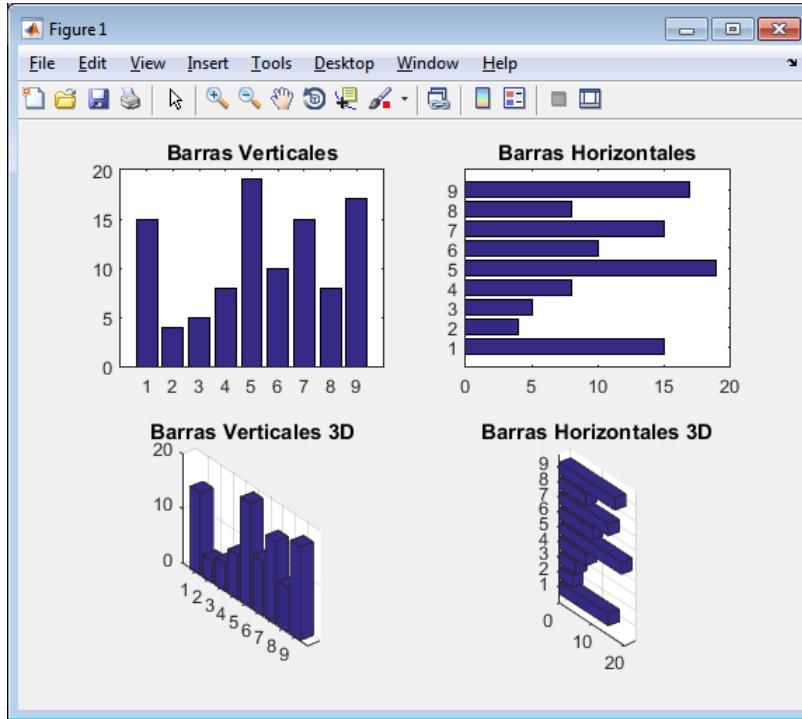
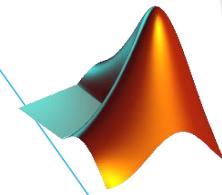
Diagrama de barras

Hay varias formas para representar diagramas de barras y estos son los comandos en Matlab.

Comando	Significado
bar(Y)	Gráfico de barras verticales relativo al vector de frecuencias Y . si Y es matriz se obtiene el grafico de barras múltiple para cada fila de Y .
bar(Y,'estilo')	Gráfico con estilo para las barras. Los estilos son ' group ' (estilo por defecto con barras verticales agrupadas) y ' stack ' (barras apiladas). Si la matriz Y es (m,n) , el grafico agrupado tiene m grupos de n barras verticales.
Barh(...)	Gráficos de barras horizontales.
Bar3(...)	Grafica de barras verticales en 3 dimensiones.
Bar3h(...)	Grafica de barras horizontales en 3 dimensiones.

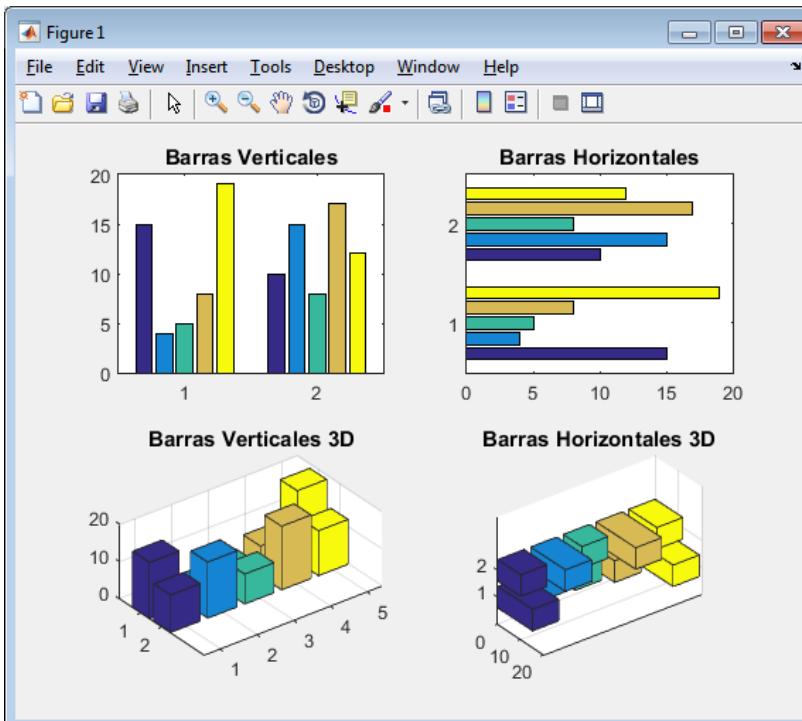
Veamos ejemplos cuando **Y** es un vector. A continuación pondremos los distintos casos en una sola ventana grafica

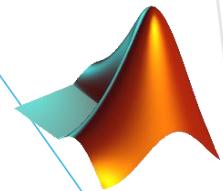
```
>> Y=[15 4 5 8 19 10 15 8 17];
>> subplot(2,2,1), bar(Y), title('Barras Verticales')
>> subplot(2,2,2), barh(Y), title('Barras Horizontales')
>> subplot(2,2,3), bar3(Y), title('Barras Verticales 3D')
>> subplot(2,2,4), bar3h(Y), title('Barras Horizontales 3D')
```



Ahora cuando Y es una matriz,

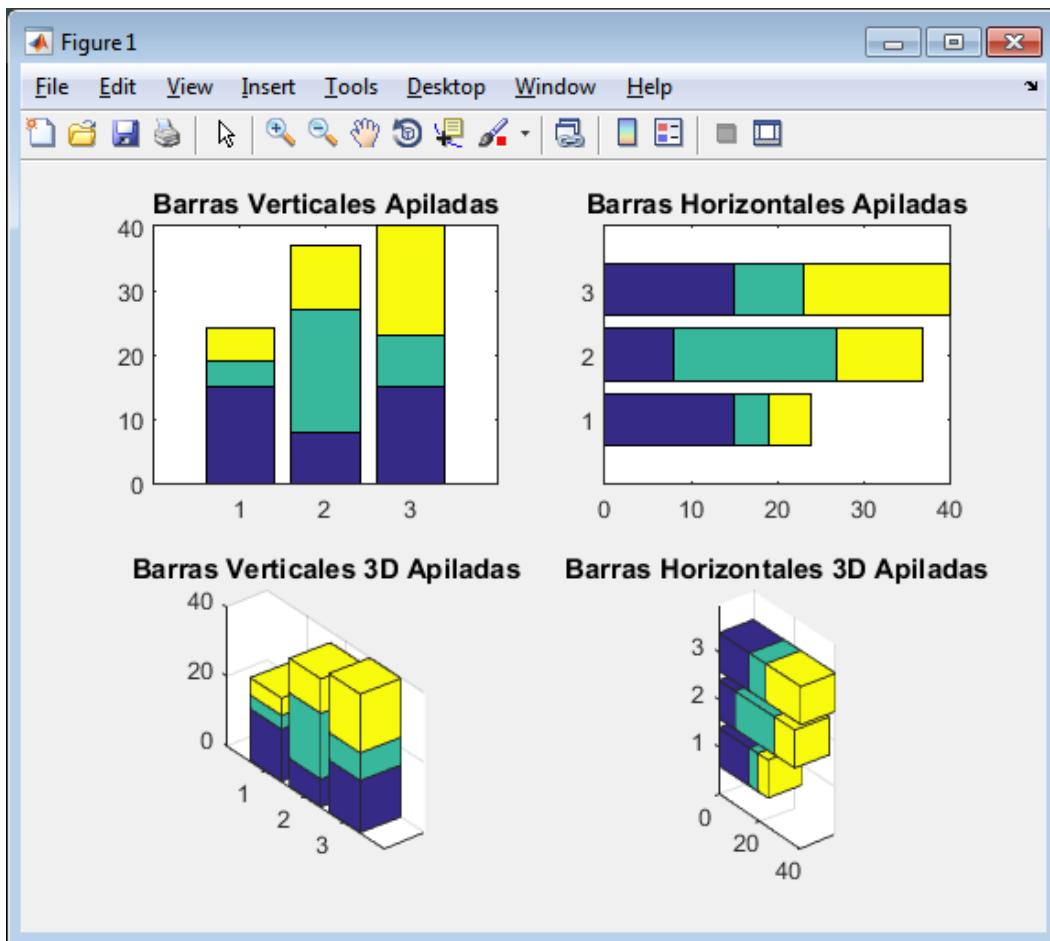
```
>> Y=[15 4 5 8 19; 10 15 8 17 12];
>> subplot(2,2,1), bar(Y), title('Barras Verticales')
>> subplot(2,2,2), barh(Y), title('Barras Horizontales')
>> subplot(2,2,3), bar3(Y), title('Barras Verticales 3D')
>> subplot(2,2,4), bar3h(Y), title('Barras Horizontales 3D')
```

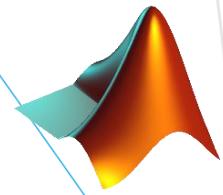




Por último, también se pueden agrupar en 3D, con la orden `bar3(x,'group')` y para que aparezcan las barras apiladas lo hacemos con el comando `bar3(x,'stack')`.

```
>> Y=[15 4 5; 8 19 10; 15 8 17];
>> subplot(2,2,1), bar(Y,'stack'), title('Barras Verticales Apiladas')
>> subplot(2,2,2), barh(Y,'stack'), title('Barras Horizontales Apiladas')
>> subplot(2,2,3), bar3(Y,'stack'), title('Barras Verticales 3D Apiladas')
>> subplot(2,2,4), bar3h(Y,'stack'), title('Barras Horizontales 3D Apiladas')
```





Histogramas

Comando	Significado
hist(Y)	Dibuja el histograma relativo al vector de frecuencias Y usando 10 rectángulos de igual base. Si Y es una matriz, se construye un histograma para cada una de las columnas.
hist(Y,x)	Dibuja el histograma relativo al vector de frecuencias Y usando tantas cajas como elementos tiene el vector x y entrando cada caja en los sucesivos valores de x .
hist(Y,k)	Dibuja el histograma relativo a Y usando tantas cajas como indica el escalar k .
[n,x]=hist(...)	Devuelve los vectores n y x con frecuencias asignadas a cada caja del histograma y los valores en los que se centran las cajas.

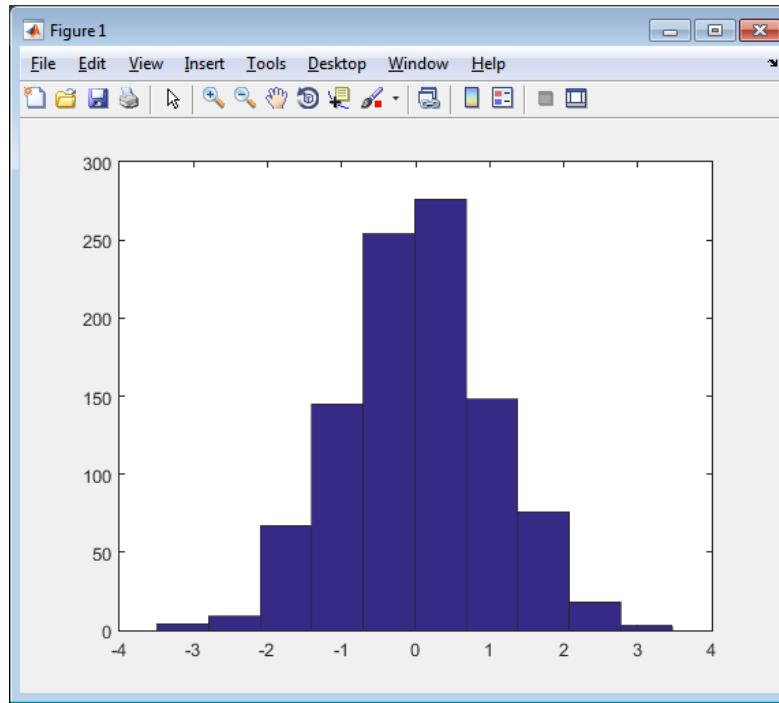
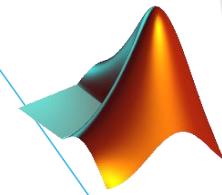
Por ejemplo para generar 1000 números aleatorios normales (0,1). Con la orden **hist(x)** se gráfica el histograma de frecuencias relativo a dichos puntos.

```
>> y=randn(1000,1);
>> hist(y)
>> [n,x]=hist(y)

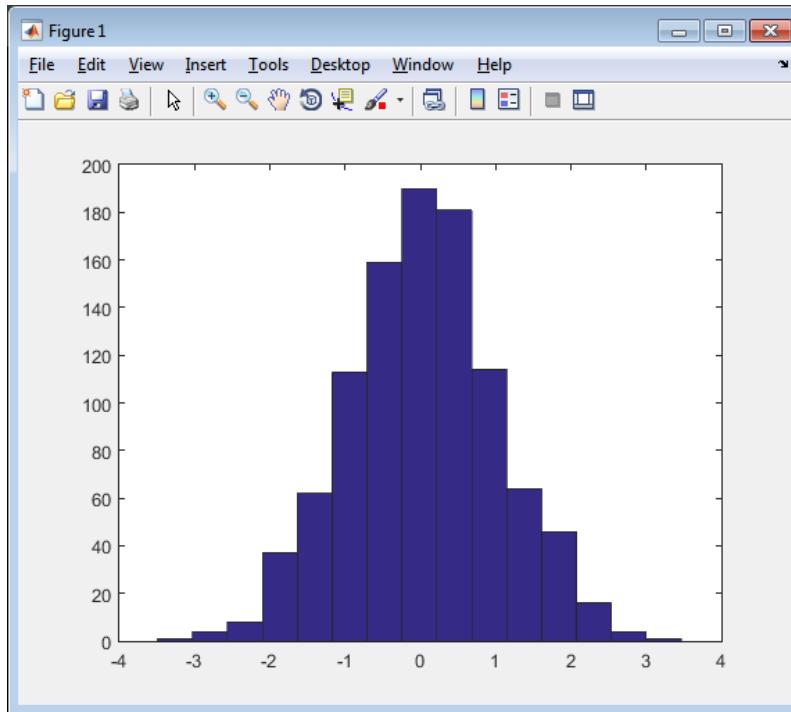
n =
4      9      67     145     254     276     148      76      18       3

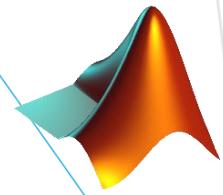
x =
Columns 1 through 7
-3.1437   -2.4479   -1.7521   -1.0563   -0.3605    0.3352    1.0310

Columns 8 through 10
1.7268    2.4226    3.1184
```



```
>> hist(y,15)
```

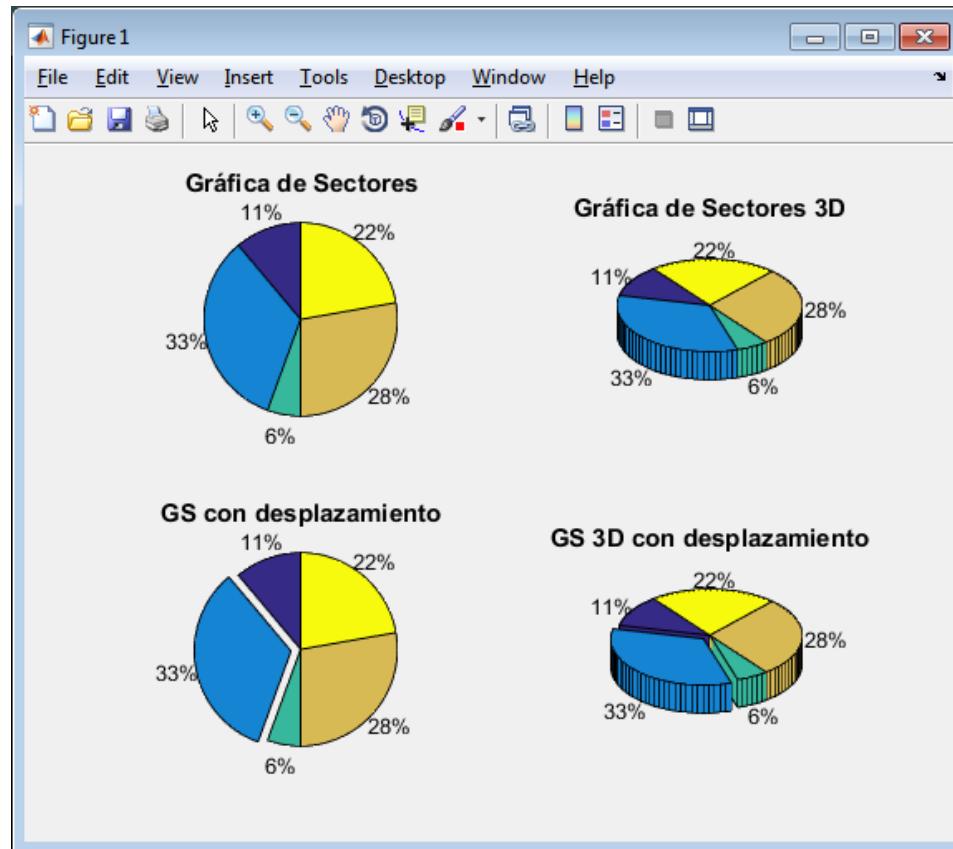


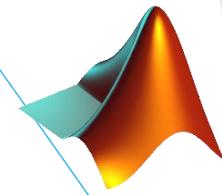


Graficas de sectores

Comando	Significado
Pie(X)	Realiza el grafico de sectores relativo al vector de frecuencias X .
Pie(X,Y)	Realiza el grafico de sectores relativo al vector de frecuencias X desplazando hacia fuera los sectores en los que $Y_i \neq 0$.
Pie3(...)	Realiza grafica de sectores en 3D.

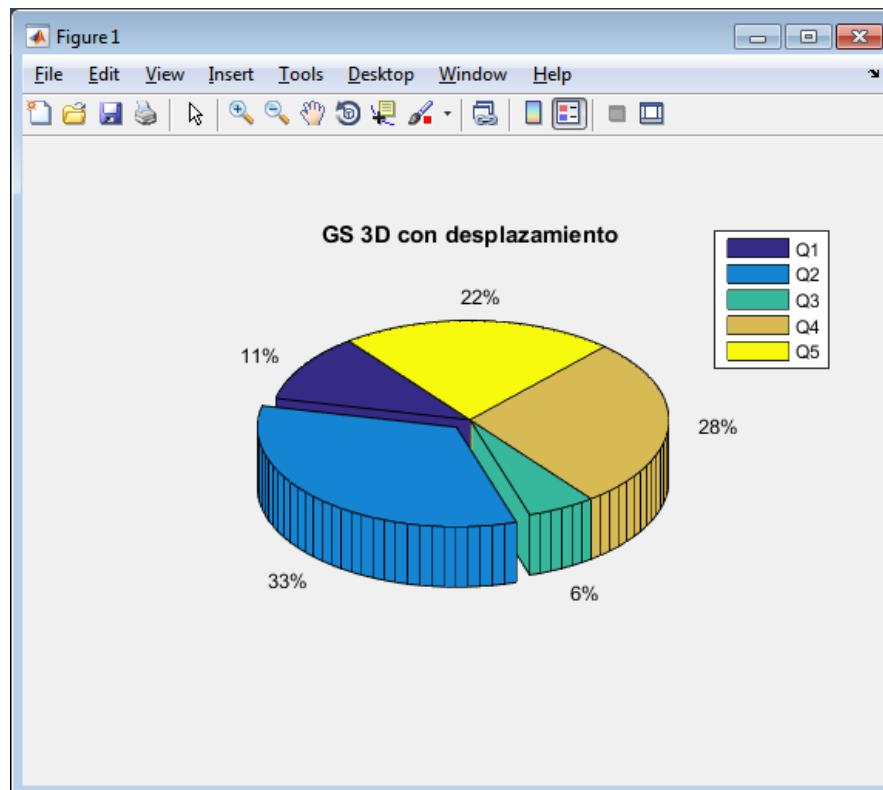
```
>> x=[1 3 0.5 2.5 2];
>> y=[0 1 0 0 0];
>> subplot(2,2,1), pie(x), title('Gráfica de Sectores')
>> subplot(2,2,2), pie3(x), title('Gráfica de Sectores 3D')
>> subplot(2,2,3), pie(x,y), title('GS con desplazamiento')
>> subplot(2,2,4), pie3(x,y), title('GS 3D con desplazamiento')
```





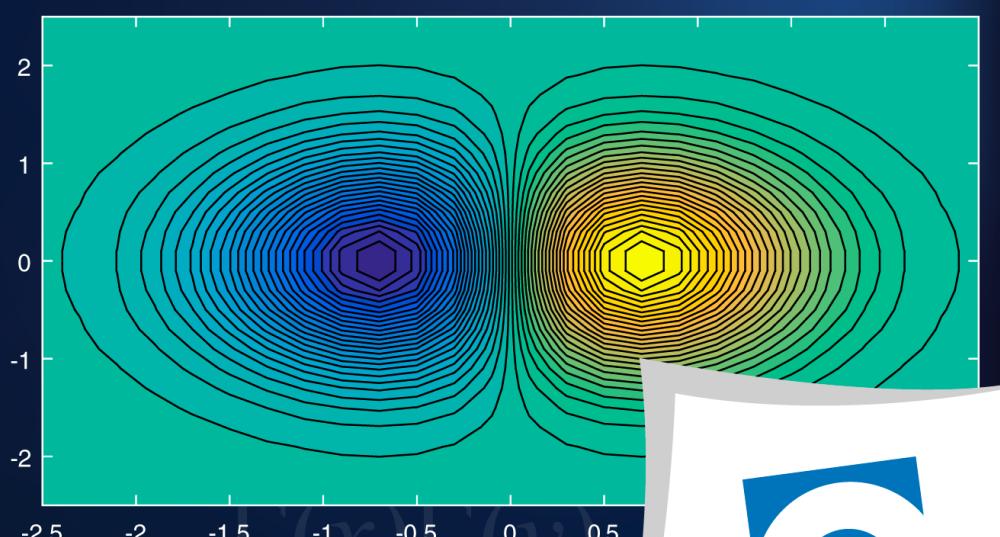
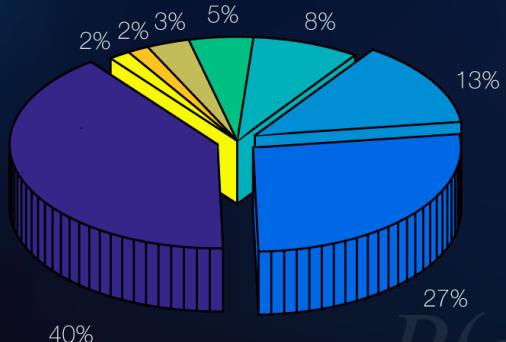
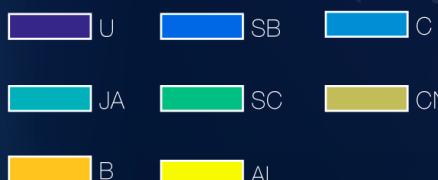
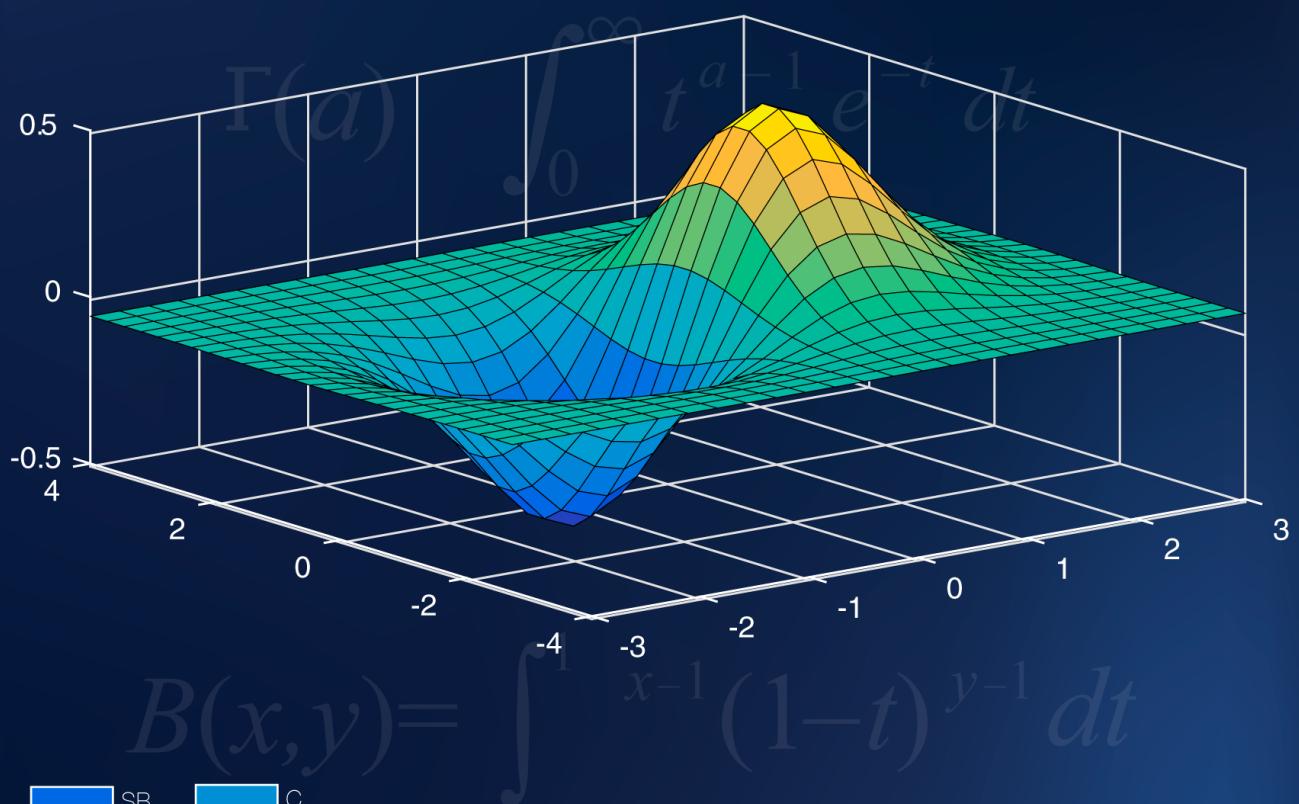
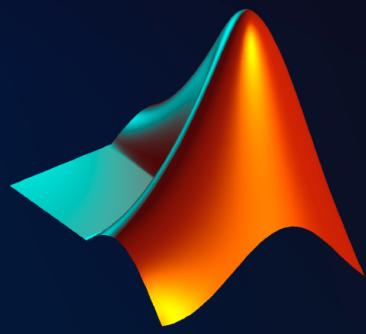
También podemos usar legendas y tener así una idea más clara de a quién pertenece un porcentaje en específico.

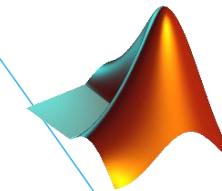
```
>> x=[1 3 0.5 2.5 2];
>> y=[0 1 0 0 0];
>> pie3(x,y), title('GS 3D con desplazamiento')
>> legend('Q1','Q2','Q3','Q4','Q5')
```



Curso MATLAB

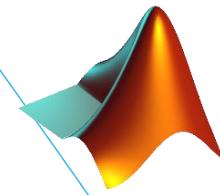
Intermedio





Contenido

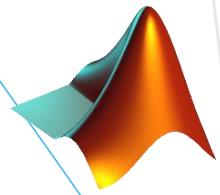
Gráficas de líneas en 3D	3
Generación de mallado en el plano xy	5
Gráficos de superficies en 3D	6
Curvas de nivel.....	8
Coordenadas cilíndricas, esféricas y paramétricas	10
Coordenadas cilíndricas	10
Coordenadas esféricas	11
Coordenadas paramétricas.....	12
Formas geométricas especiales	13
Cilindros, esferas y elipses.....	13



Sesión 3: graficas tridimensionales

Gráficas de líneas en 3D

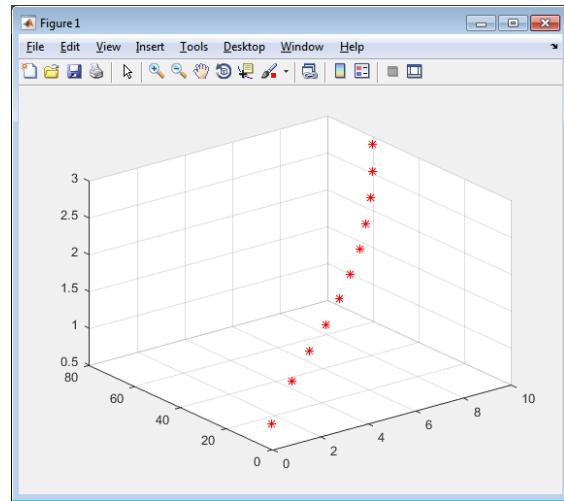
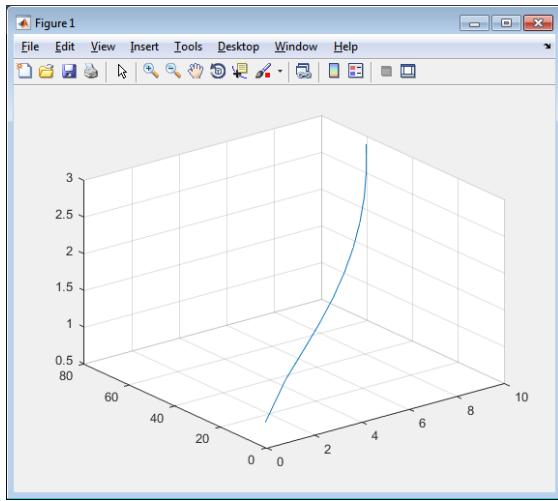
Plot3(x,y,z)	Dibuja el conjunto de puntos (x,y,z) , donde x, y, z son vectores fila x, y, z pueden ser coordenadas paramétricas o matrices de la misma dimensión, en cuyo caso se hace una gráfica por cada tripleta de filas y sobre los mismos ejes. Para valores complejos de x, y, z se ignoran las partes imaginarias.
Plot3(x,y,z,s)	Grafica de plot(x,y,z) con las opciones definidas en S se compone de dos dígitos entre comillas simples, el primero de los cuales fija el color de la línea del gráfico y el segundo el carácter a usar en el graficado. Los valores posibles de colores y caracteres se han descrito ya al explicar el comando plot .
Plot3(x1,y1,z1,s1 ,x2,y2,z2,s2,...)	Combina sobre los mismos ejes, los gráficos definidos para las tripletas (X_i, Y_i, Z_i, S_i) . Se trata de una forma de representar varias funciones sobre el mismo gráfico.



Ejemplo: Plot3(x,y,z), Plot3(x,y,z,s)

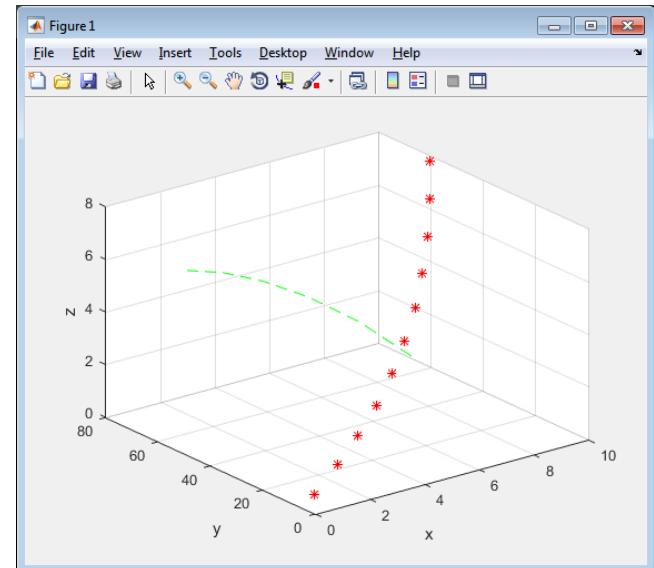
```
>> x=0:10;
>> y=0.5*(x+1).^2;
>> z=(x+y).^(1/4);
>> plot3(x,y,z)
```

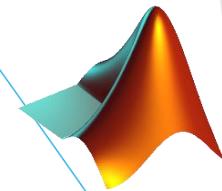
```
>> x=0:10;
>> y=0.5*(x+1).^2;
>> z=(x+y).^(1/4);
>> plot3(x,y,z,'r*')
>> grid
```



Ejemplo: Plot3(x1,y1,z1,s1,x2,y2,z2,s2,...)

```
>> x1=0:10;
>> y1=0.5*(x1+1).^2;
>> z1=(x1.^2+y1.^2).^(1/4);
>> x2=0:5;
>> y2=50-(x2+1).^2;
>> z2=(x2.^2+y2.^2).^(1/4);
>> plot3(x1,y1,z1,'r*',x2,y2,z2,'g--')
>> grid
>> xlabel('x'), ylabel('y'), zlabel('z')
```





Generación de mallado en el plano xy

Meshgrid(x,y), Transforma el campo de definición dado de las variables **x** e **y** de la función a representar **z=f(x,y)** en argumento matricial por el comando **mesh** para obtener el grafico de malla.

```
>> x=[1 3 5]; y=[2 4 6 8];
>> [X,Y]=meshgrid(x,y)
```

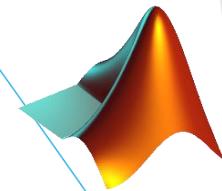
X =

```
1      3      5
1      3      5
1      3      5
1      3      5
```

Y =

```
2      2      2
4      4      4
6      6      6
8      8      8
```

(1,8)	(3,8)	(5,8)
(1,6)	(3,6)	(5,6)
(1,4)	(3,4)	(5,4)
(1,2)	(3,2)	(5,2)

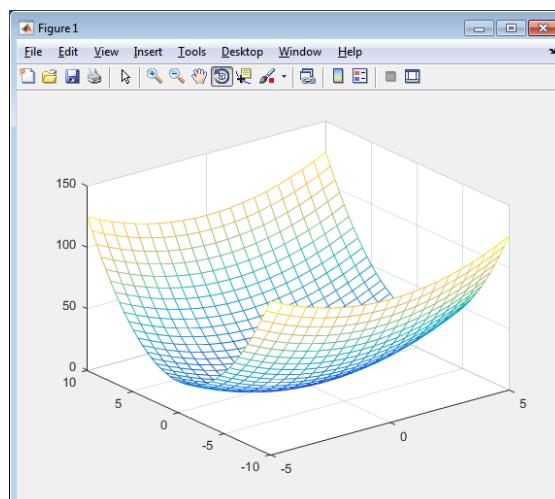


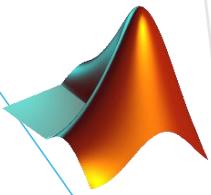
Gráficos de superficies en 3D

Comando	Significado
mesh(X,Y,Z,C)	Representa el gráfico de malla de la función $z=f(x,y)$, dibujando las líneas de la rejilla que componen la malla con los colores especificados en C (C puede ser ignorado).
meshz(X,Y,Z,C)	Representa el grafico de malla de la función $z=f(x,y)$ con una especie de cortina o telón en la parte inferior.
meshc(X,Y,Z,C)	Representa el grafico de malla de la función $z=f(x,y)$ con el grafico de contorno correspondiente (curvas de nivel proyectadas sobre el plano XY).
surf(X,Y,Z)	Representa el grafico de superficie de la función $z=f(x,y)$, realizando el dibujo con los colores especificados en C (C puede ser ignorado).
surfl(X,Y,Z,C)	Representa el grafico de superficie de la función $z=f(x,y)$, realizando el dibujo sombreado.

Ejemplo: `mesh(X,Y,Z,C)`

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> mesh(X, Y, Z)
```

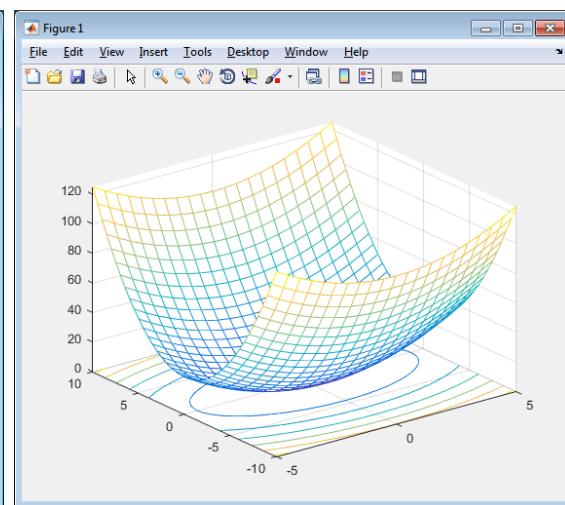
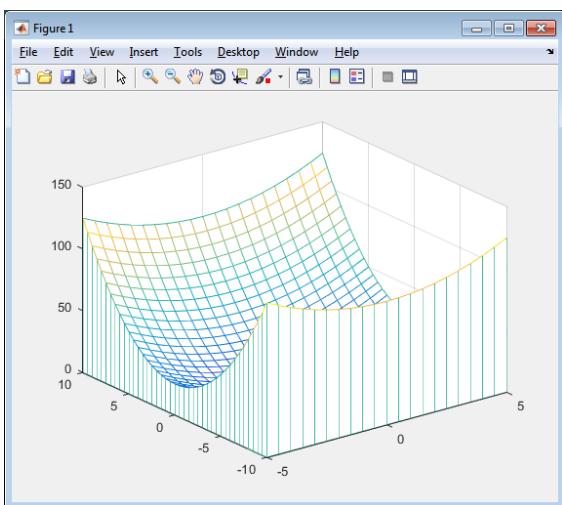




Ejemplo: meshz(X,Y,Z), meshc(X,Y,Z)

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> meshz(X, Y, Z)
```

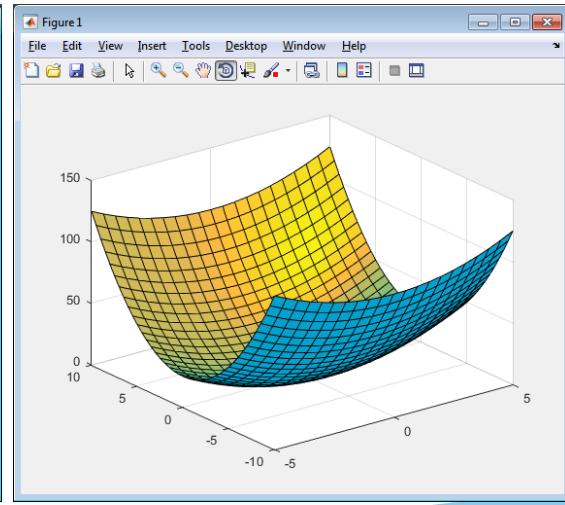
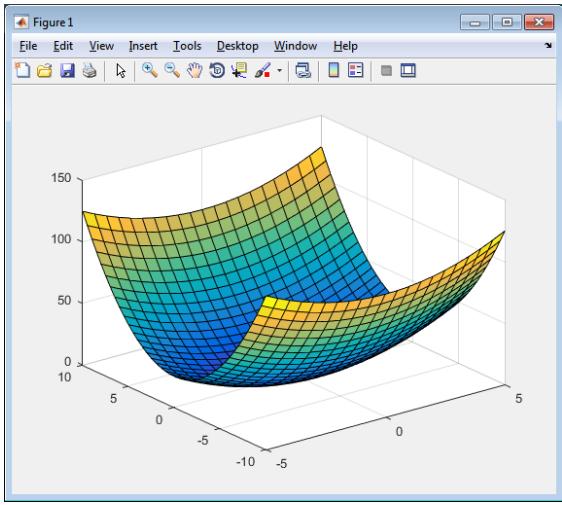
```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> meshc(X, Y, Z)
```

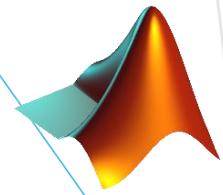


Ejemplo: surf(X,Y,Z), surfl(X,Y,Z)

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> surf(X, Y, Z)
```

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> surfl(X, Y, Z)
```





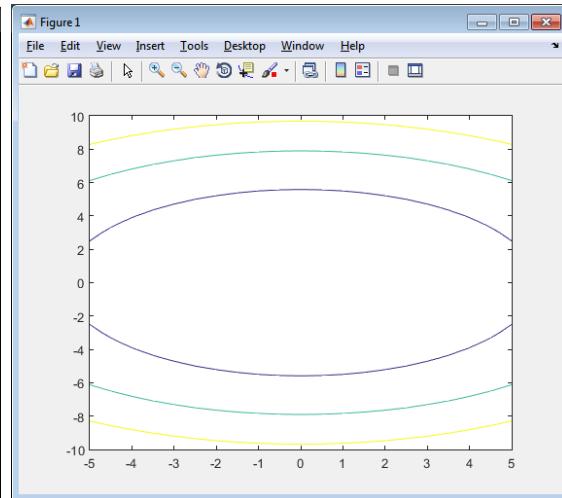
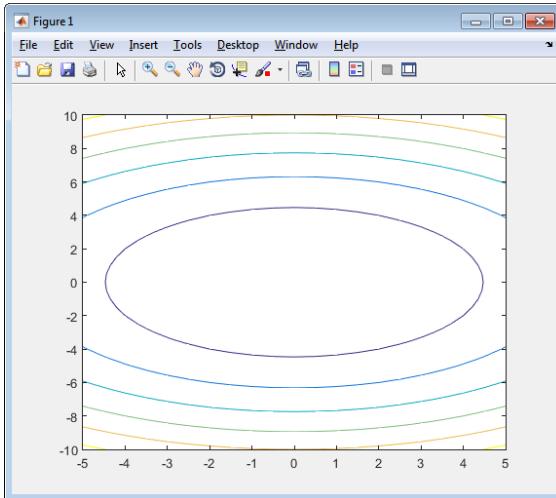
Curvas de nivel

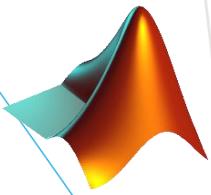
Comandos	Resultado
contour(X,Y,Z,n) contour(X,Y,Z)	Dibuja el grafico de contorno (curvas de nivel) para las coordenadas (X,Y,Z) ; usando n líneas de contorno. Si obviamos n el número de líneas es automático.
contour3(X,Y,Z,n) contour3(X,Y,Z)	Dibuja los gráficos de contorno en 3 dimensiones con n curvas. Si obviamos n el número de líneas es automático.
pcolor(X,Y,Z)	Dibuja un gráfico de contorno (curvas de nivel) para la matriz (X,Y,Z) utilizando una representación basada en densidades de colores. Suele denominarse gráfico de densidad.

Ejemplos: `contour(X,Y,Z)`, `contour(X,Y,Z,n)`

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> contour (X, Y, Z)
```

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X, Y]=meshgrid(x, y);
>> Z=X.^2+Y.^2;
>> contour (X, Y, Z, 3)
```

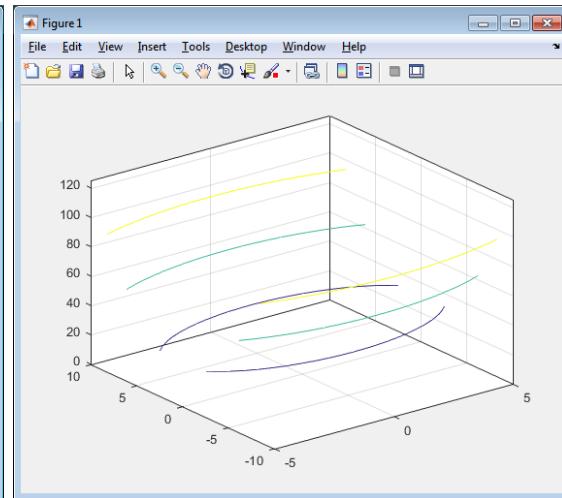
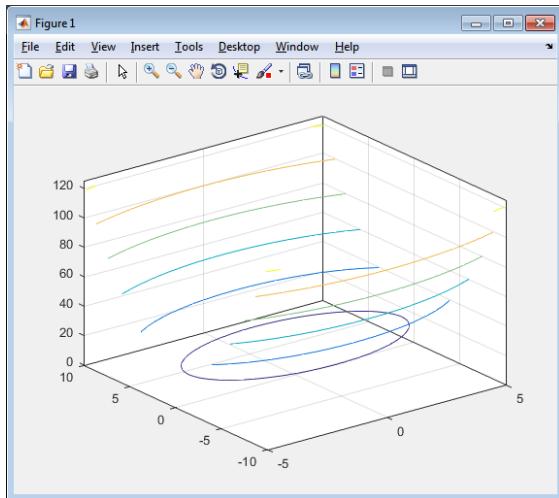




Ejemplos: `contour3(X,Y,Z)`, `contour3(X,Y,Z,n)`

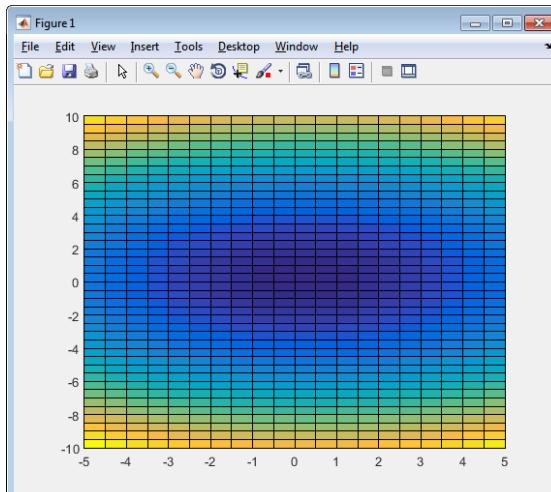
```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2+Y.^2;
>> contour3(X,Y,Z)
```

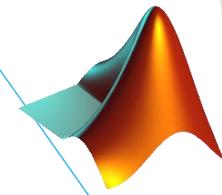
```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2+Y.^2;
>> contour3(X,Y,Z,3)
```



Ejemplo: `pcolor(X,Y,Z)`

```
>> x=-5:0.5:5;
>> y=-10:0.5:10;
>> [X,Y]=meshgrid(x,y);
>> Z=X.^2+Y.^2;
>> pcolor(X,Y,Z)
```





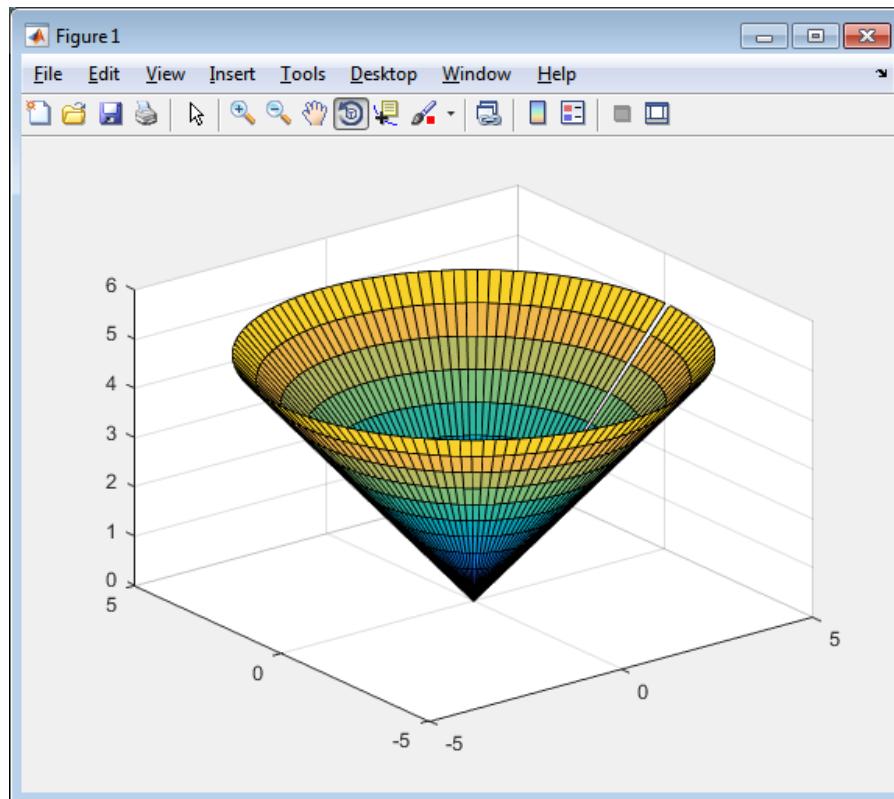
Coordenadas cilíndricas, esféricas y paramétricas

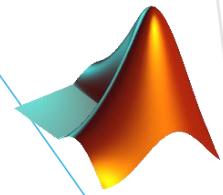
Coordenadas cilíndricas

$$\begin{cases} x = r \cos(u); \\ y = r \sin(u); & u \in [\alpha, \beta], r \in [r_1, r_2] \\ z = z; \end{cases}$$

Ejemplo: graficar el cono $z = \sqrt{x^2 + y^2}$ en el dominio circular de radio 6 centrado en el origen de coordenadas.

```
>> [r,u]=meshgrid(0:0.5:5,0:0.05:2*pi);
>> x=r.*cos(u);
>> y=r.*sin(u);
>> z=sqrt(x.^2+y.^2);
>> surf(x,y,z)
```





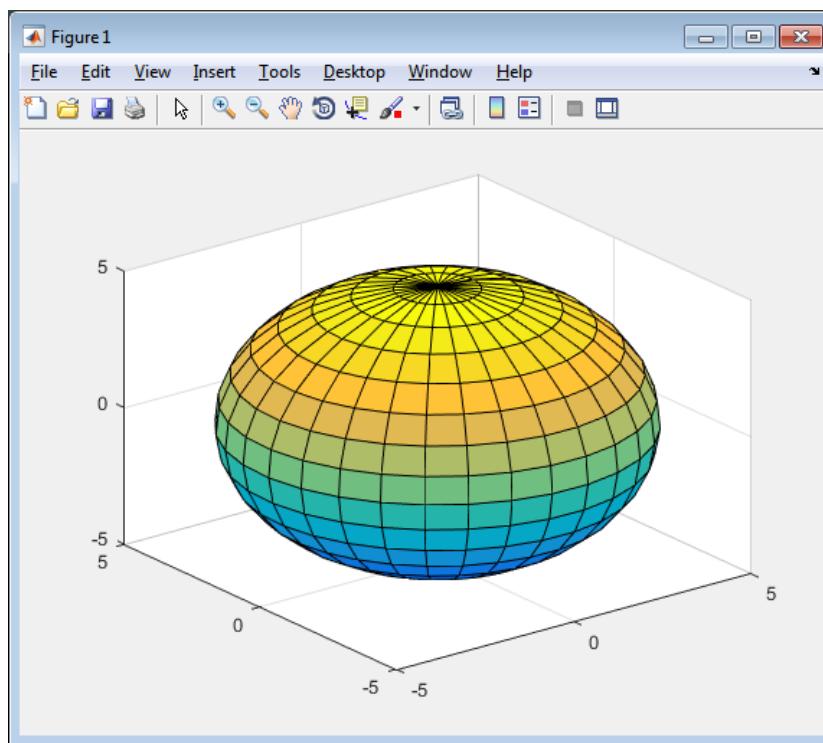
Coordenadas esféricas

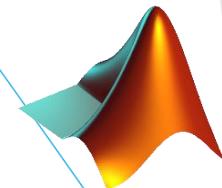
$$\begin{cases} x = r \sin(v) \cos(u); \\ y = r \sin(v) \sin(u); \quad u \in [\alpha_1, \alpha_2], v \in [\theta_1, \theta_2] \\ z = r \cos(v); \end{cases}$$

Ejemplo: graficar la esfera $x^2 + y^2 + z^2 = r^2$ de radio $r=6$ en coordenadas esféricas, parametrizando en coordenadas esféricas tenemos

$$\begin{cases} x = 5 \sin(v) \cos(u); \\ y = 5 \sin(v) \sin(u); \quad u \in [0, 2\pi], v \in [0, \pi] \\ z = 5 \cos(v); \end{cases}$$

```
>> [u,v]=meshgrid(0:0.2:2*pi,0:0.2:pi);
>> x=5*sin(v).*cos(u);
>> y=5*sin(v).*sin(u);
>> z=5*cos(v);
>> surf(x,y,z)
```





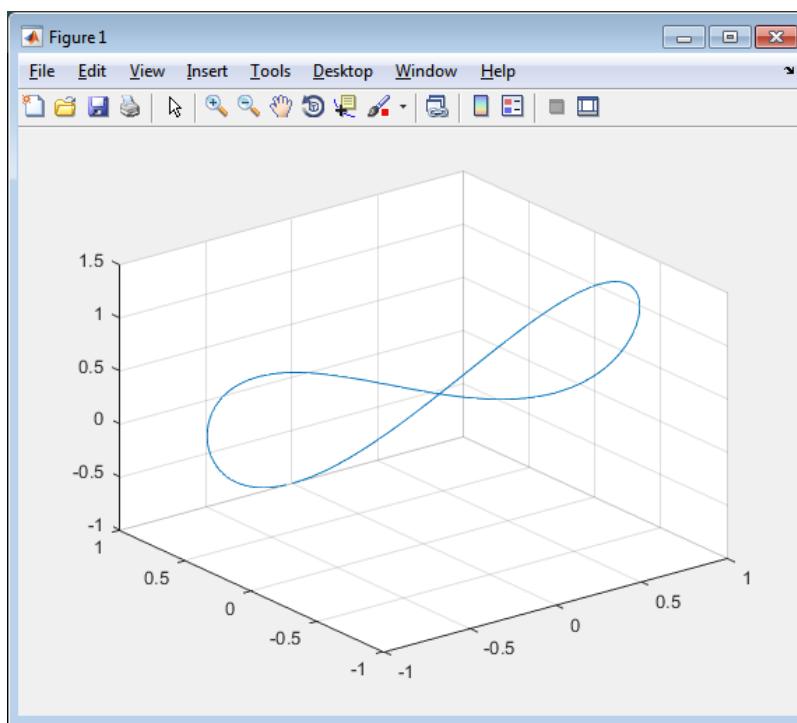
Coordenadas paramétricas

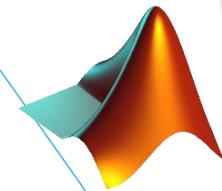
$$\begin{cases} x = x(t) \\ y = y(t) \\ z = z(t) \end{cases}$$

Ejemplo: Graficar la siguiente curva paramétrica.

$$\begin{cases} x = \sin(t); \\ y = \cos(t); & t \in [t_1, t_2] \\ z = \sin(t)^2 - \cos(t); \end{cases}$$

```
>> t=0:0.05:2*pi;
>> x=sin(t);
>> y=cos(t);
>> z=sin(t).^2 - cos(t);
>> plot3(x,y,z)
>> grid
```





Formas geométricas especiales

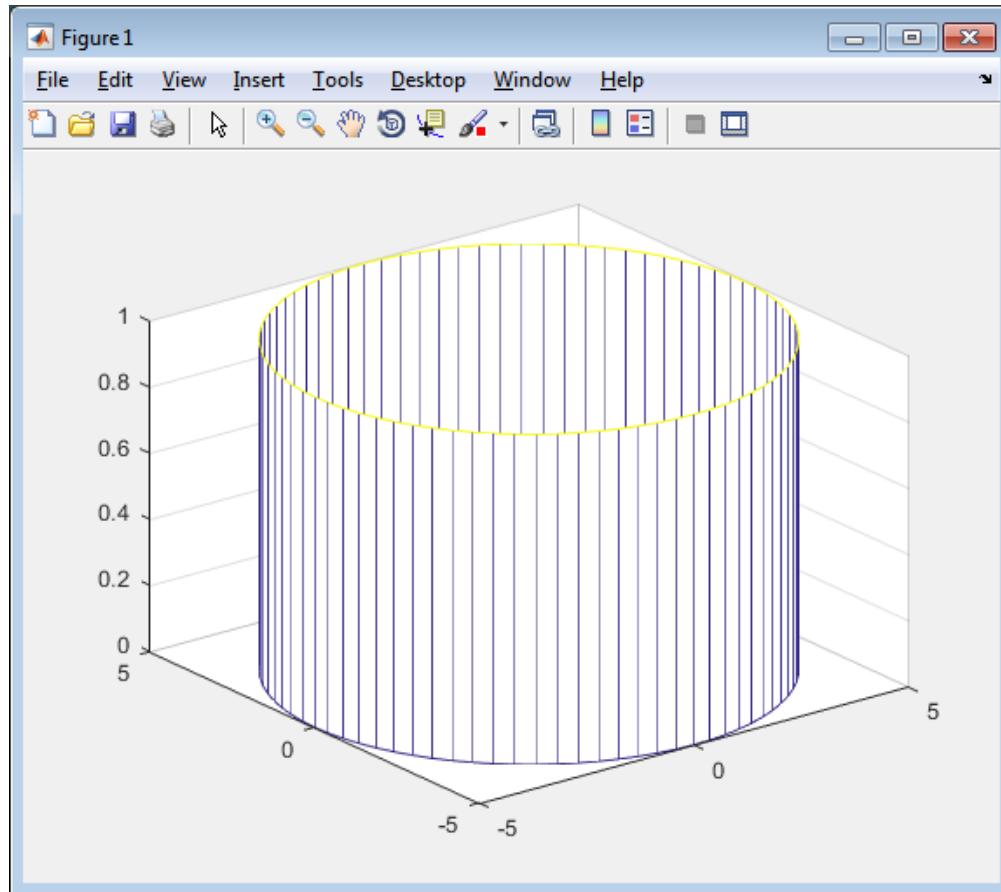
Cilindros, esferas y elipses

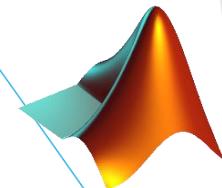
[x,y,z]=cylinder(R,n)

Retorna las coordenadas **x**, **y**, **z** necesarias para la generación de un cilindro con altura 1, radio **R** y número de puntos en cada circunferencia de **n** (20 por defecto).

Ejemplo:

```
>> [x,y,z]=cylinder(5,80);  
>> mesh(x,y,z)
```



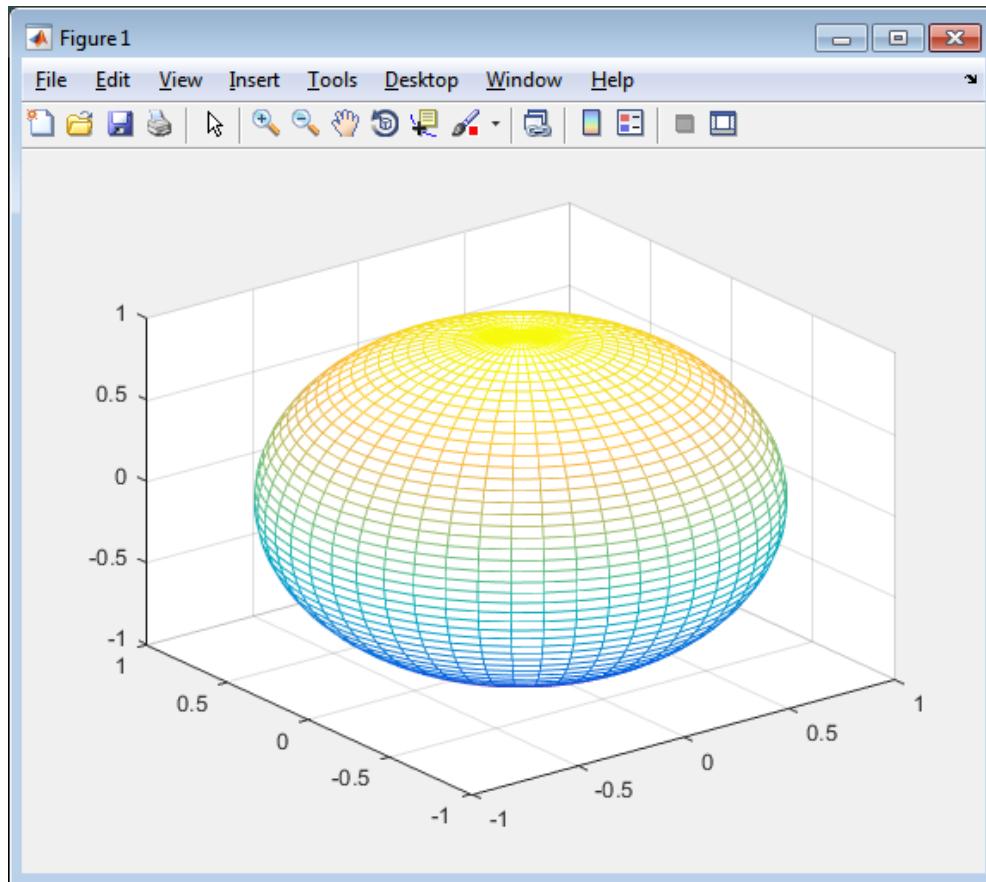


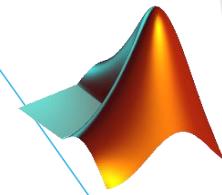
[x,y,z]=sphere(R,n)

Retorna las coordenadas **x,y,z** necesarias para la generación de una esfera con radio 1. El número de ternas es **(n+1)x(n+1)** (**n=20** por defecto).

Ejemplo:

```
>> [x,y,z]=sphere(50);  
>> mesh(x,y,z)
```



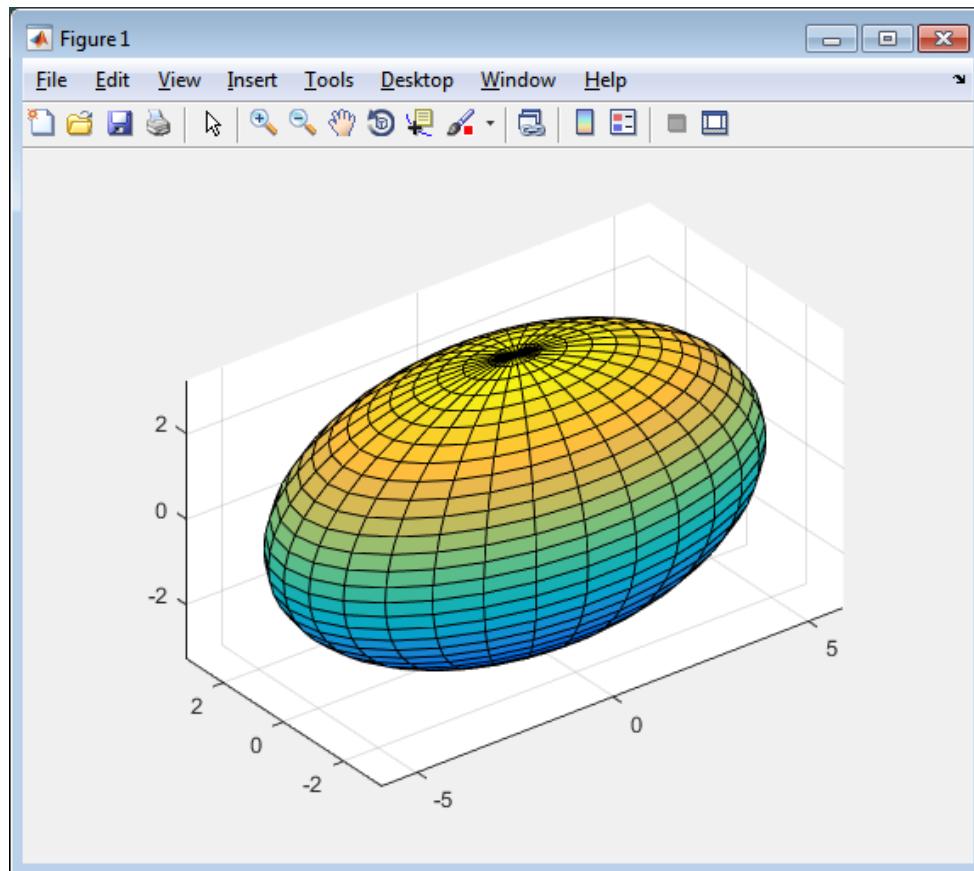


[x,y,z]=ellipsoid(xc,yc,zc,xr,yl,zr,n)

Retorna las coordenadas **x, y, z** necesarias para la generación de una elipsoide centrada en **(xc,yc,zc)** con semiejes **xr,yl,zr**. El número de ternas es **(n+1)x(n+1)** (**n=20** por defecto).

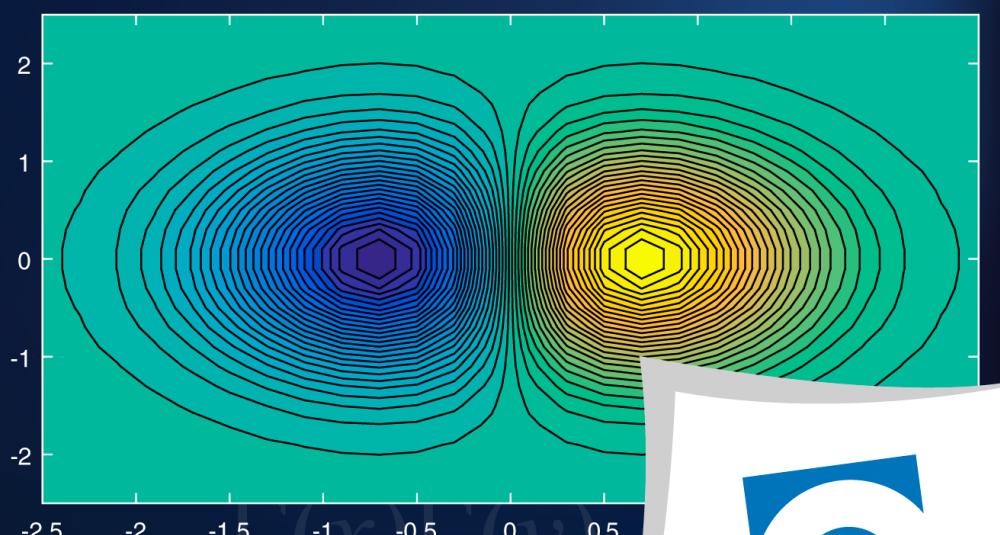
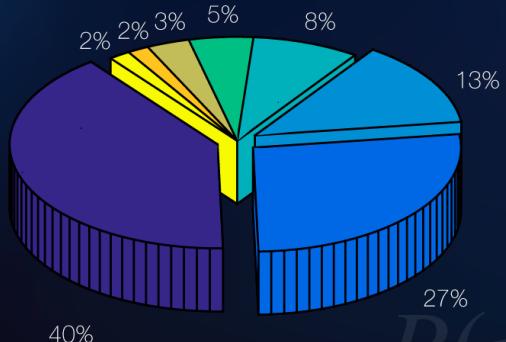
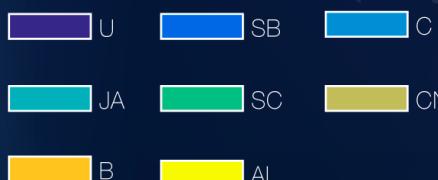
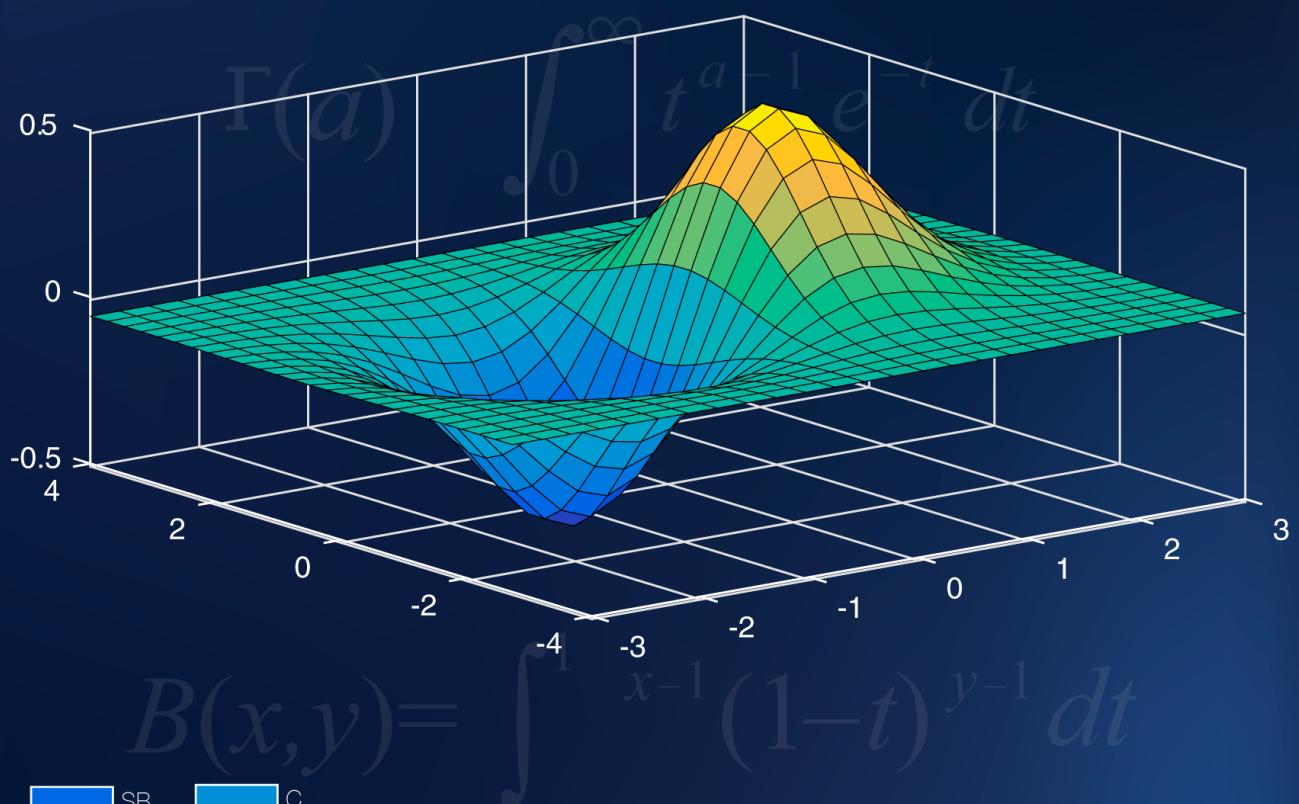
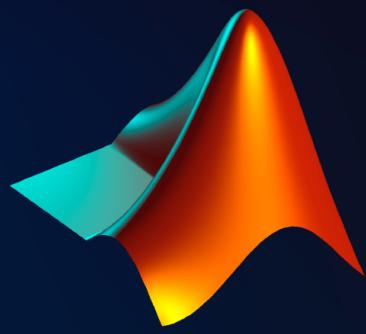
Ejemplo:

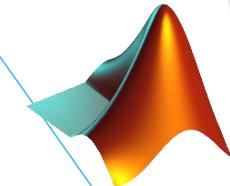
```
>> [x, y, z] = ellipsoid(0,0,0,5.9,3.25,3.25,30);  
>> surf(x,y,z)  
>> axis equal
```



Curso MATLAB

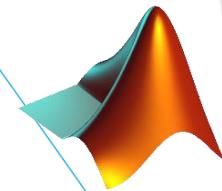
Intermedio





Contenido

Tipos de archivos de Matlab: script y function	3
Como crear, guardar y ejecutar un archivo .m.....	4
Menús de Matlab	7
Ayuda para programas creados por usuario	8
Operadores lógicos y relacionales	9
Instrucciones importantes.....	10

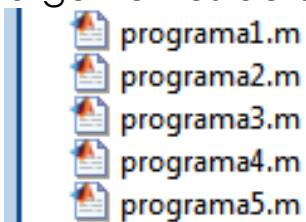


Sesión 4: Programación en Matlab

Tipos de archivos de Matlab: script y function

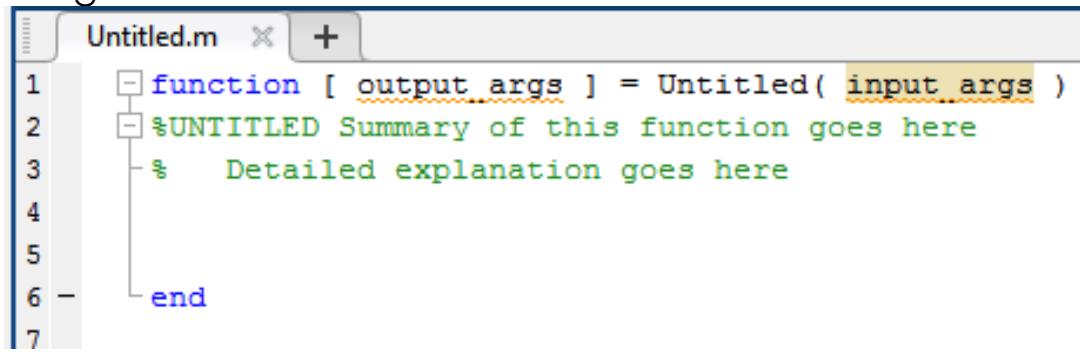
Script

Un script se define mediante un m-fichero, el cual está formado por un conjunto de sentencias pero no tiene la cualidad de ser una función como por ejemplo $y=\sin(x)$ que posee argumentos de entrada y salida.



Function

Una function o función se define mediante un m-fichero, cuyo nombre coincide con el nombre de la función. En la primera línea ejecutable debe encontrarse `function`, como en la imagen.



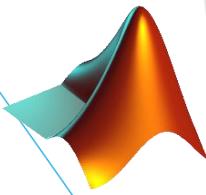
```
Untitled.m × +  
1 function [ output_args ] = Untitled( input_args )  
2 %UNTITLED Summary of this function goes here  
3 % Detailed explanation goes here  
4  
5  
6 - end  
7
```

Donde:

`Output_args`: argumentos de salida.

`Untitled`: sin título (aquí va el nombre de la función).

`Input_args`: argumentos de entrada.

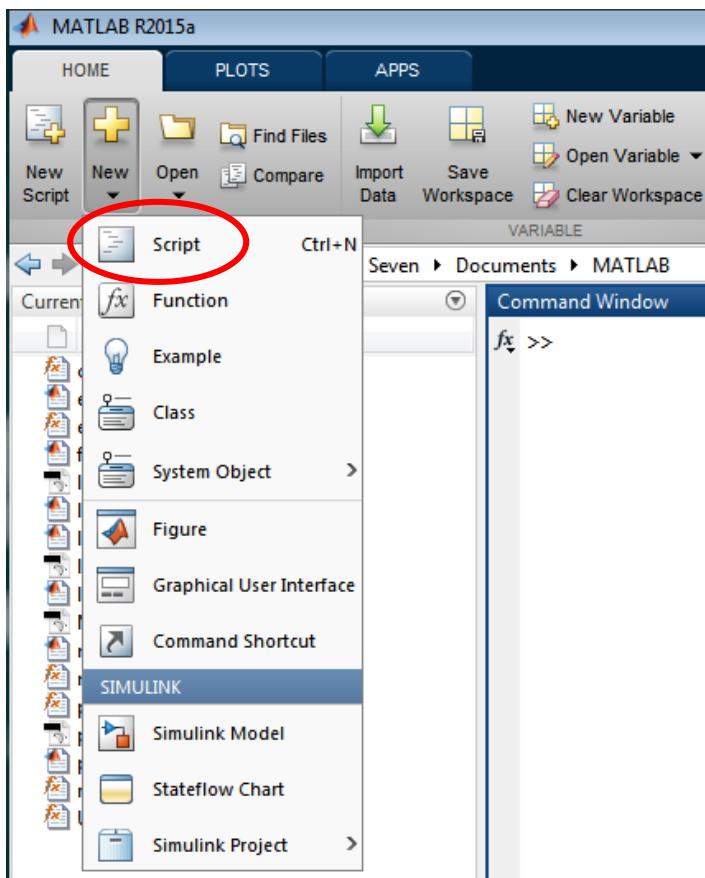


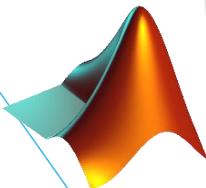
Cuando hay más de un argumento de salida estos deben ir entre corchetes y separados por comas [].

Las variables definidas en la función (salvo los argumentos) son locales. Para que el valor de una variable sea compartido por varias funciones se emplea la instrucción **global**, cuya sintaxis es **global variable** y debe aparecer en todas las funciones que la comparten.

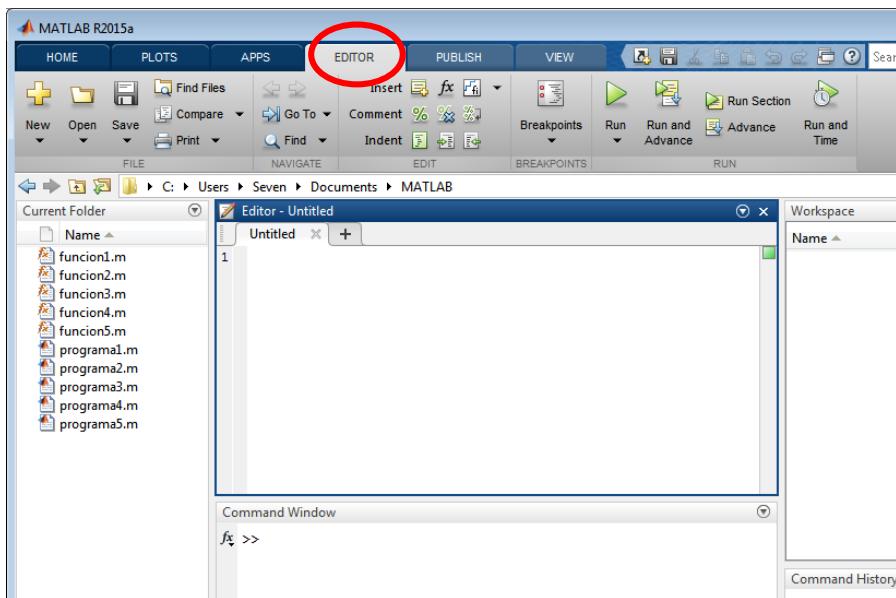
Como crear, guardar y ejecutar un archivo .m

A continuación conoceremos como Matlab nos permite crear un archivo **.m**, si se desea crear un archivo nuevo, debemos ir a la barra de herramientas, luego al ícono “**new**”, con lo cual se despliega una lista con distintas opciones escogemos la opción “**script**”, acá es donde se hacen los programas.

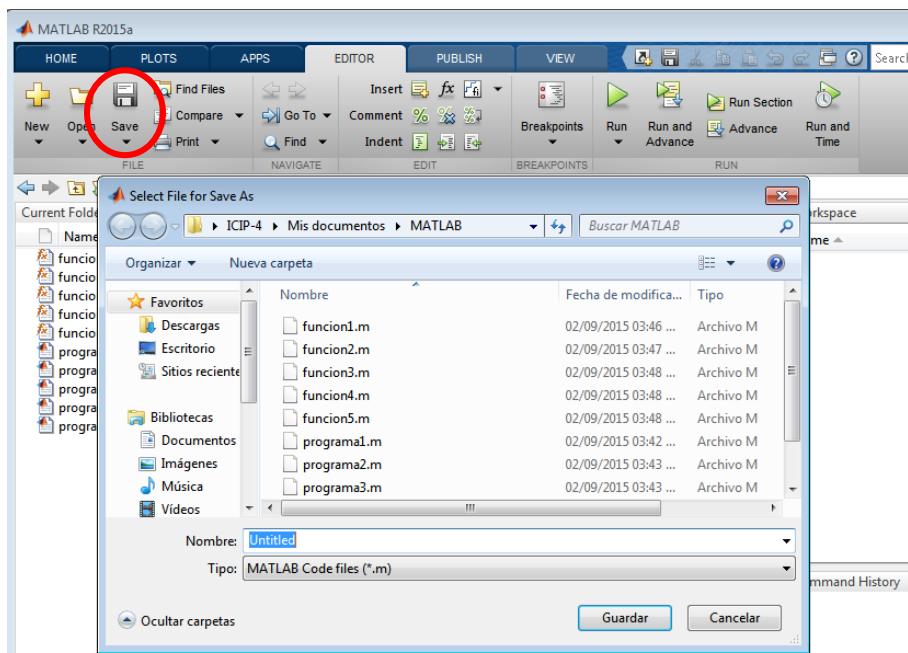


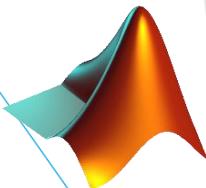


al abrir o crear un script se habilita una nueva pestaña en la barra de herramientas con el nombre de **editor** el cual muestra múltiples opciones para estos archivos, editor ofrece las facilidades también de grabación, recuperación, creación de nuevos archivos, impresión, entre otros.

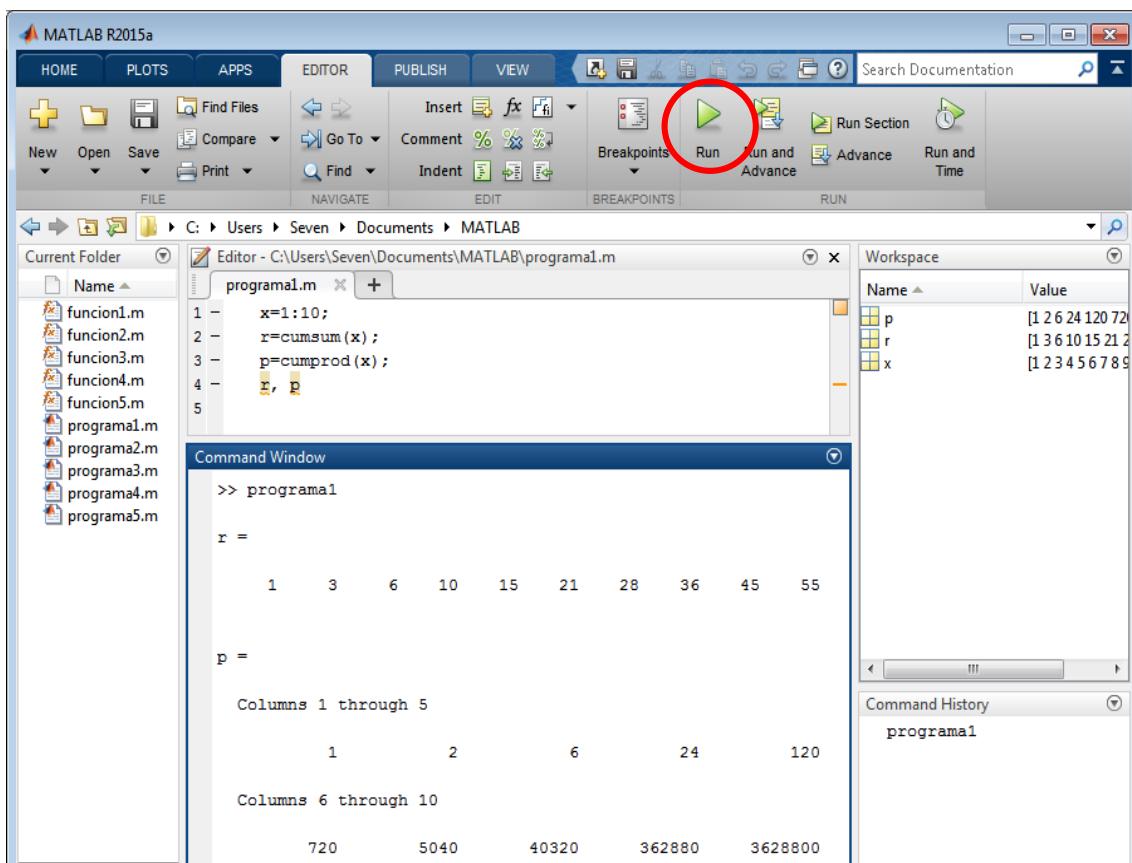


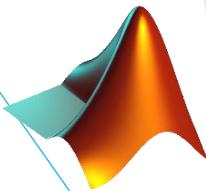
Para guardar damos clic en el botón “**save**”, el cual abrirá una ventana en la cual asignaremos el nombre y ubicación.





Supongamos que hemos creado un programa podemos ejecutarlo de distintas maneras, por ejemplo escribiendo en la línea de comandos el nombre del programa y luego ejecutándolo con **Enter**. Otra manera es manteniendo abierto el programa y en editor presionar el botón “**Run**” y también podemos ejecutar el programa presionando la tecla **f5**.





Menús de Matlab

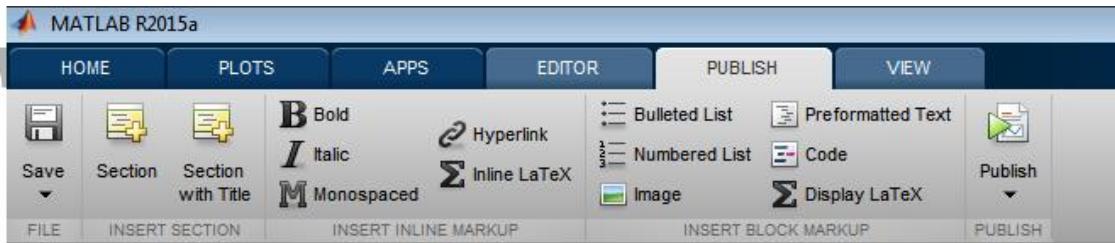
Como vimos al trabajar con los archivos .m se habilitan 3 pestañas más en la barra de herramientas: editor, publish y view. Cada una con distintas opciones

Editor

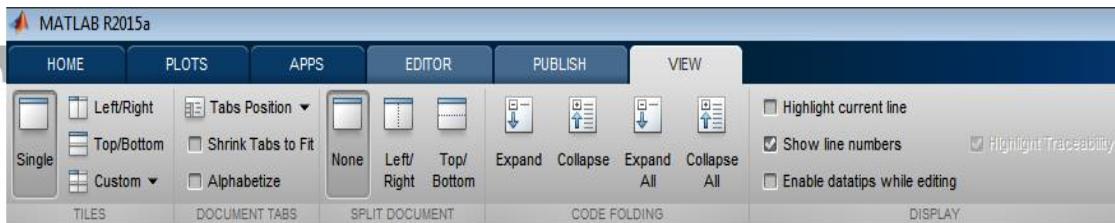
Aquí se muestran las opciones relacionadas a la creación, edición y ejecución de los archivos, como lo que acabamos de ver

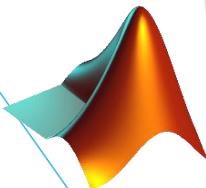


Publish



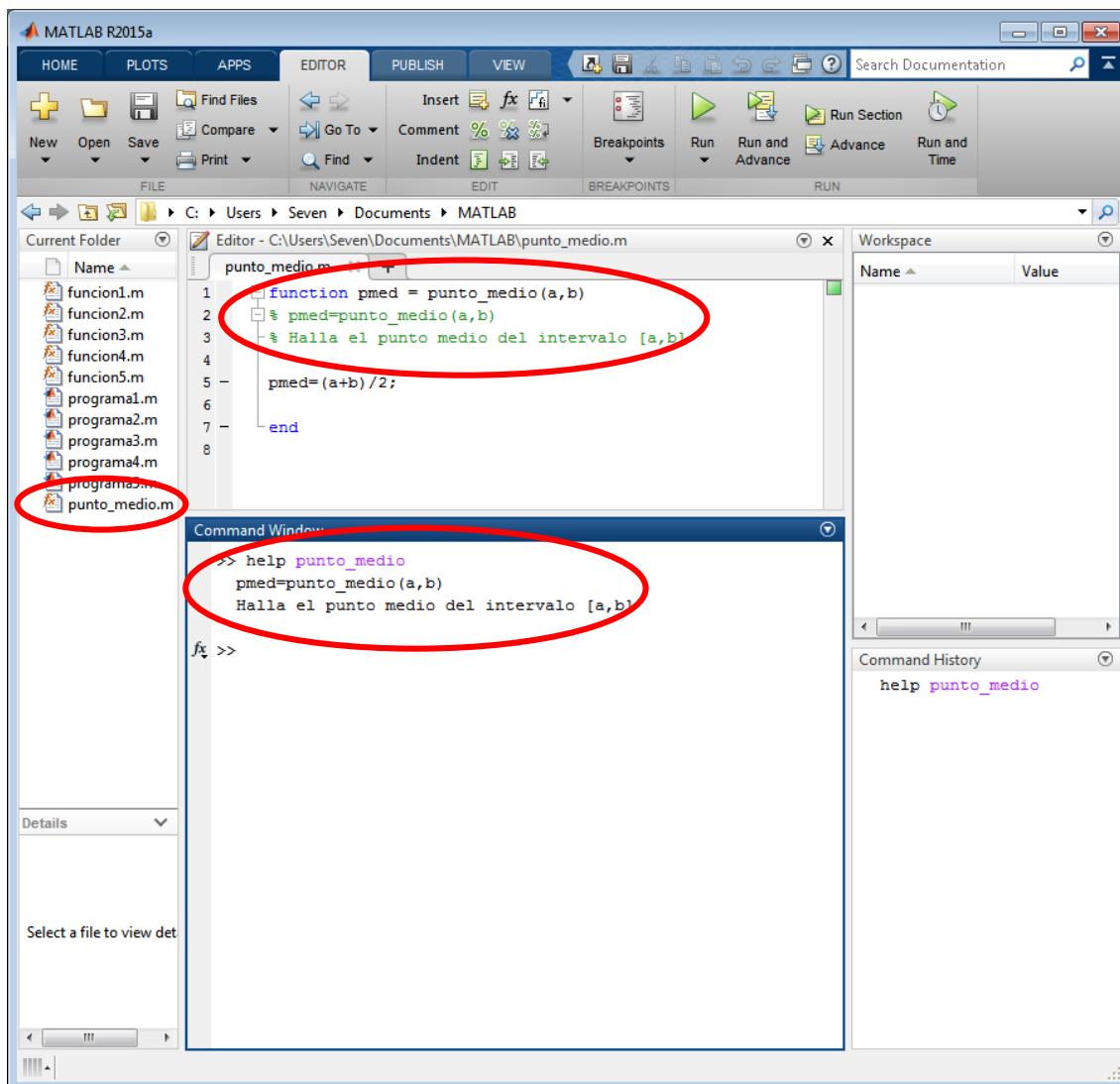
View

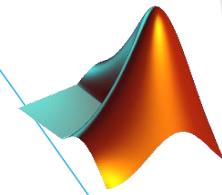




Ayuda para programas creados por usuario

Es conveniente poner un comentario en la primera línea de los programas que creamos y más que sea algo relevante, esto anteponiendo el carácter porcentaje (%) antes de empezar con el comentario; y para ver la ayuda del programa se digita en la ventana de comandos de la siguiente manera: **help nombre_funcion**. Veamos con el ejemplo siguiente.





Operadores lógicos y relacionales

Como entradas a todas las expresiones relacionales y lógicas, Matlab considera que cualquier número distinto de cero es verdadero, y es falso si es igual a cero. La salida 1 es verdadero, y cero si es falso.

Operadores Lógicos:

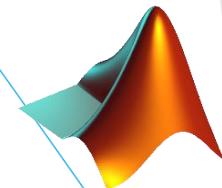
La salida de las operaciones lógicas se puede utilizar también en operaciones matemáticas.

Los operadores lógicos proporcionan un medio de combinar o negar expresiones relacionales.

Símbolo	Operador Lógico
\sim	Negación lógica o complementario (NOT)
$\&$	Conjunción lógica o intersección (AND)
$ $	Disyunción lógica (OR) o unión de A y B
$Xor(A,B)$	OR exclusivo

Operadores Relacionales:

Símbolo	Operador Relacional
$<$	Menor que
\leq	Menor que o igual a
$>$	Mayor que
\geq	Mayor que o igual a
$=$	Igual a
\neq	Distinto de



Instrucciones importantes

Return: La función puede finalizar en cualquier punto utilizando la instrucción `return`, y retorna al sitio donde ha sido invocada.

Nargin: Número de argumentos de entrada que el usuario ha pasado a la función.

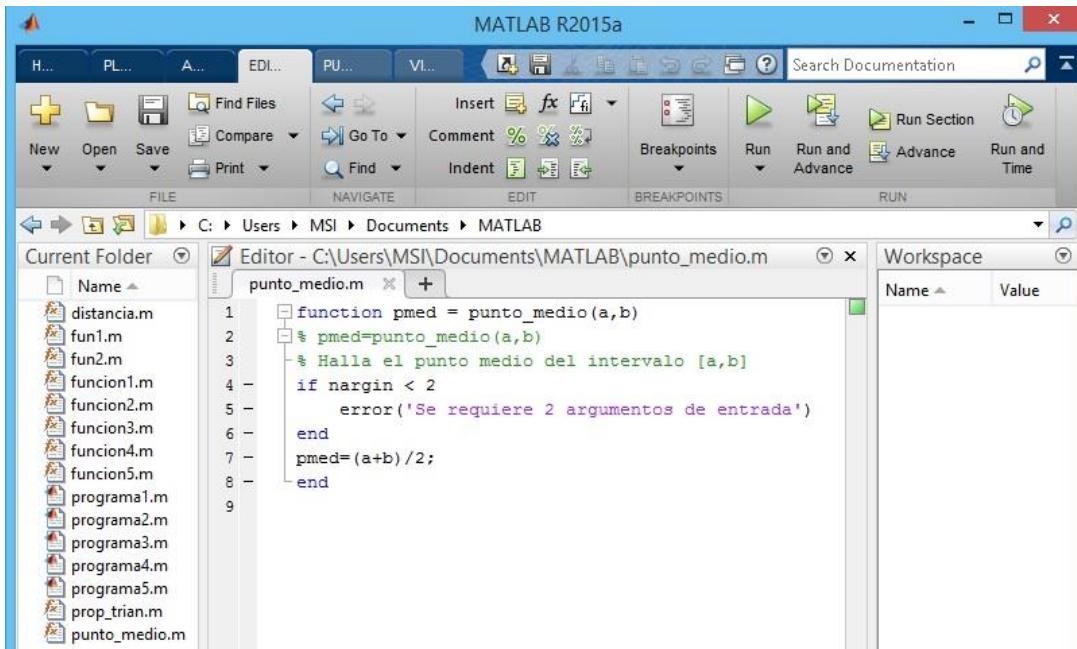
Nargout: Número de argumentos de salida que el usuario ha pasado a la función.

Error('message') muestra un mensaje de error y finaliza al programa.

Error('message',a1,a2,...) muestra un mensaje de error contenido en formato similar al `printf` de Matlab y finaliza el programa.

Ejemplo:

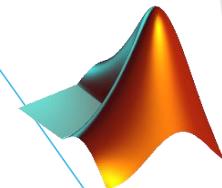
El siguiente programa halla el punto medio del intervalo **[a,b]** y muestra el error en la función **punto_medio(a,b)** cuando hay menos de 2 argumentos.



```

function pmed = punto_medio(a,b)
% pmed=punto_medio(a,b)
% Halla el punto medio del intervalo [a,b]
if nargin < 2
    error('Se requiere 2 argumentos de entrada')
end
pmed=(a+b)/2;
end

```



Veamos primero la ejecución en la ventana de comandos y luego la ayuda del programa.

Command Window

```
>> pmed=punto_medio()
Error using punto_medio (line 5)
Se requiere 2 argumentos de entrada

>> pmed=punto_medio(10)
Error using punto_medio (line 5)
Se requiere 2 argumentos de entrada

>> help punto_medio
pmed=punto_medio(a,b)
Halla el punto medio del intervalo [a,b]

>> pmed=punto_medio(10,16)

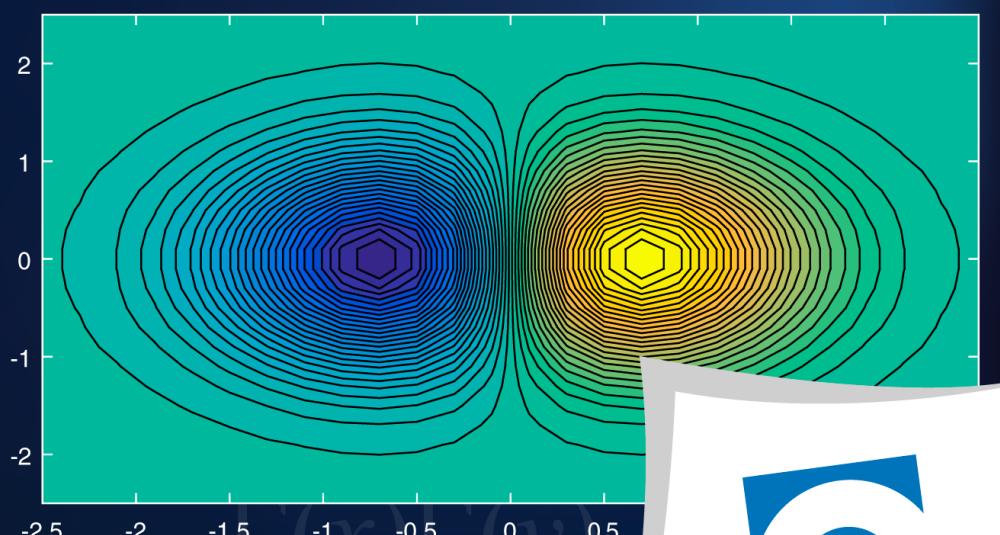
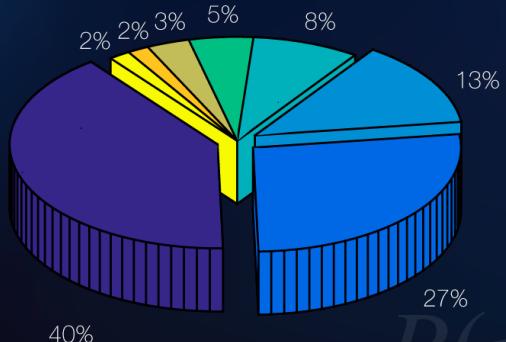
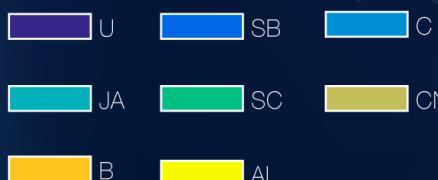
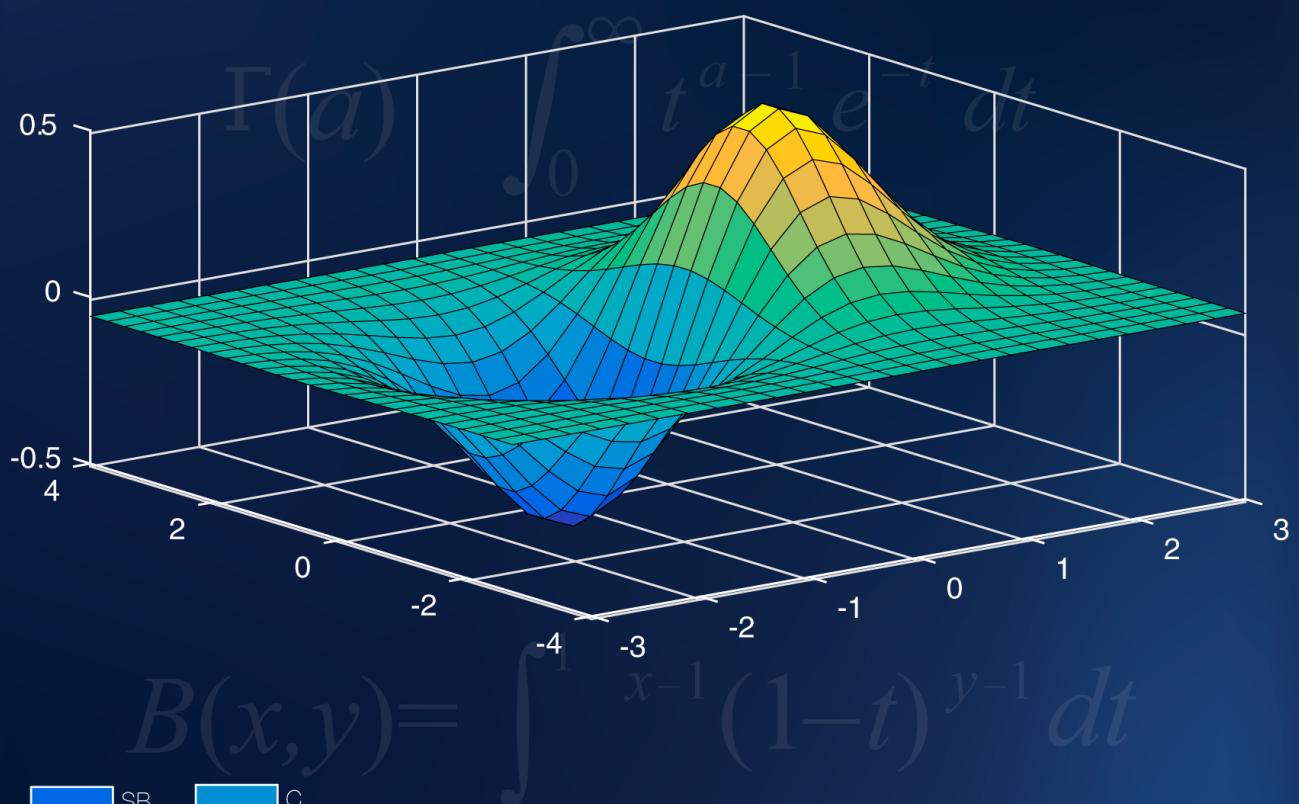
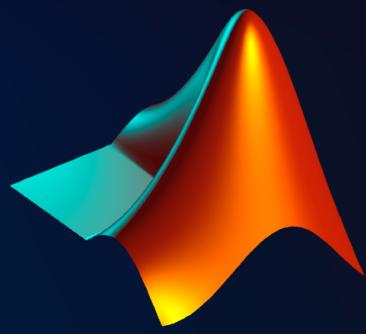
pmed =

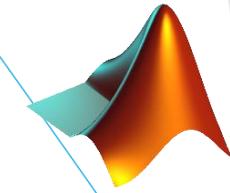
    13

fx >> |
```

Curso MATLAB

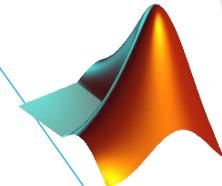
Intermedio





Contenido

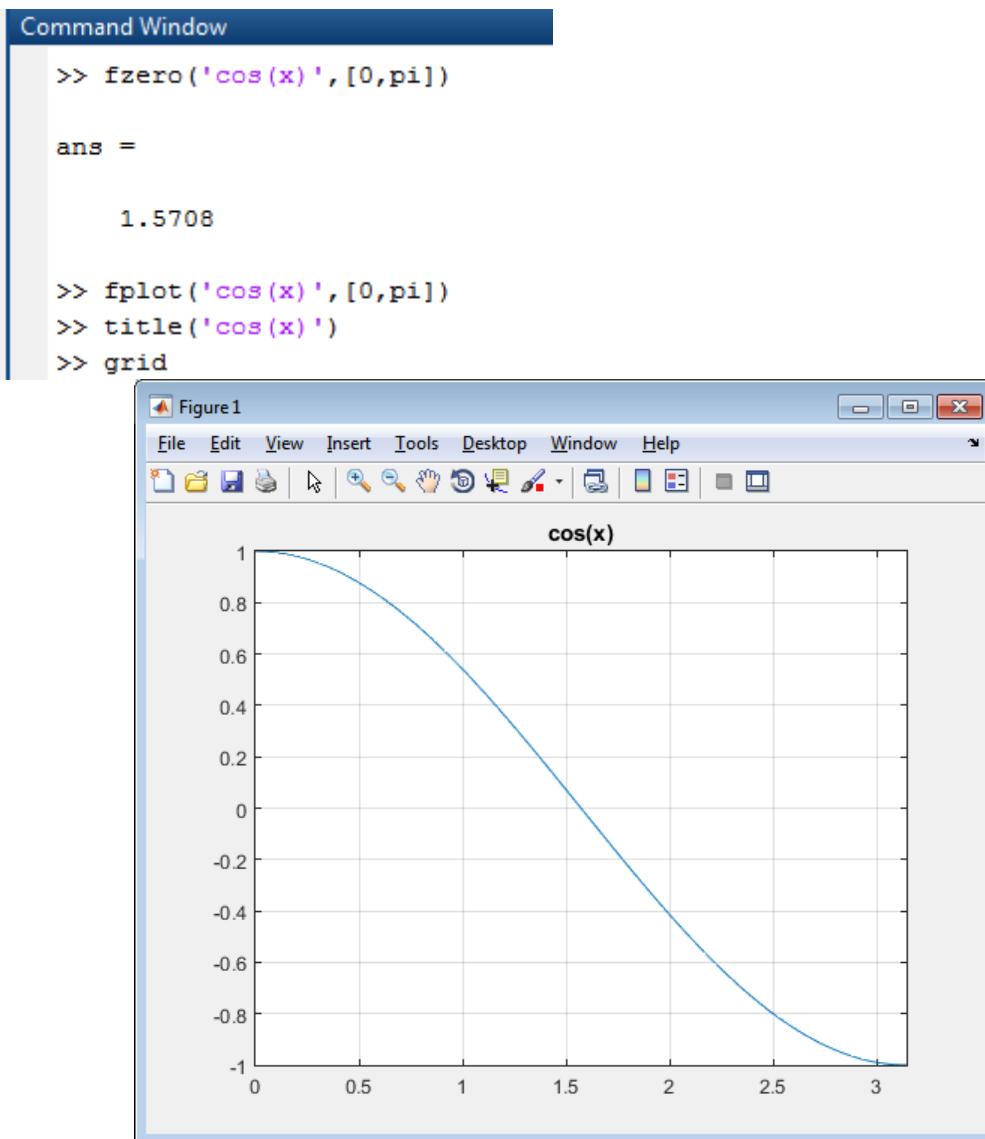
Raíces de funciones	3
Creación de funciones.....	4
Evaluación de funciones.....	5
Problemas resueltos	9

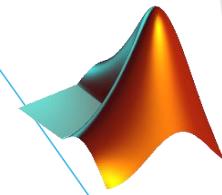


Sesión 5: Funciones

Raíces de funciones

La función **fzero()**, sirve para hallar la raíz de una función. Su sintaxis es **x=fzero(fun,[a b])**, donde **fun** es la función de la cual queremos hallar su raíz y **[a,b]** es el intervalo donde queremos hallar la raíz.





Creación de funciones

Comando **inline()** :

Sirve para crear archivos-m tipo **function**. Su sintaxis es la siguiente:

f=inline('expresión'): convierte en función simbólica la cadena '**expresión**'.

f=inline('expresion', arg1, arg2,...,argn): convierte en función simbólica a la cadena '**expresión**' con **n** argumentos de entrada de datos.

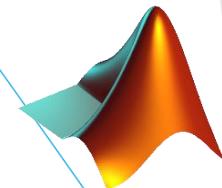
Ejemplo:

Creación de una función simbólica en una variable

```
>> f=inline('x*cos(x)+2*x^2')  
  
f =  
  
    Inline function:  
    f(x) = x*cos(x)+2*x^2
```

Creación de una función simbólica de dos variables

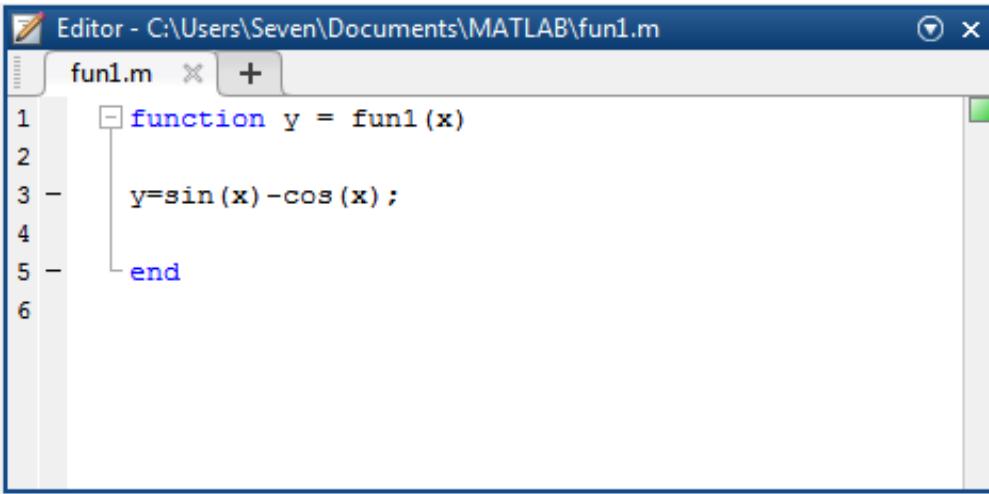
```
>> f=inline('exp(x*y)-x*sin(3*y)', 'x', 'y')  
  
f =  
  
    Inline function:  
    f(x,y) = exp(x*y)-x*sin(3*y)
```



Evaluación de funciones

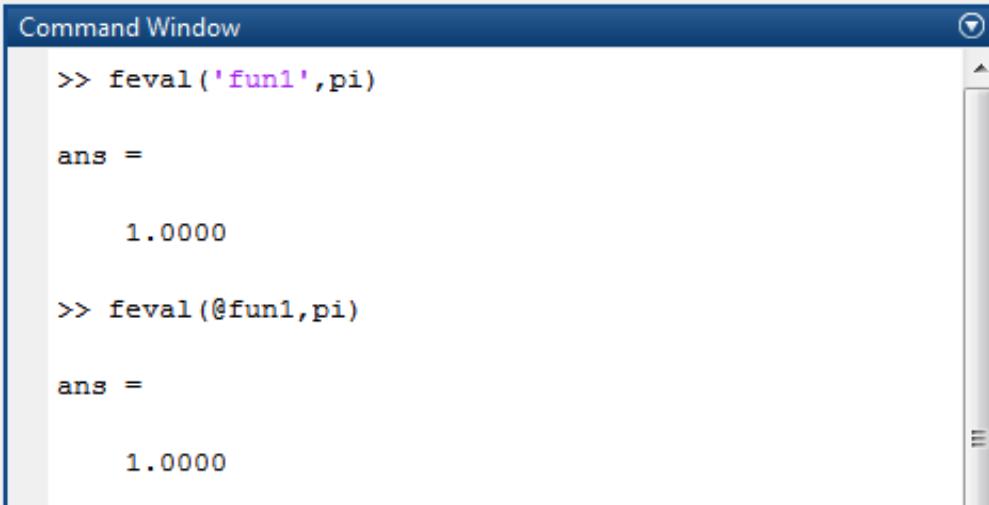
Comando feval(): Sirve para evaluar funciones. Su sintaxis es la siguiente. **feval('fun',arg1,...,argn)** o **feval(@fun,arg1,...,argn)**, Donde **fun** es una función que queremos evaluar.

Ejemplo 1: Creamos una función **fun1.m** en una sola variable en el editor de texto.



```
Editor - C:\Users\Seven\Documents\MATLAB\fun1.m
fun1.m + 
1 function y = fun1(x)
2
3 y=sin(x)-cos(x);
4
5 end
6
```

Ejecutamos en la ventana de comandos

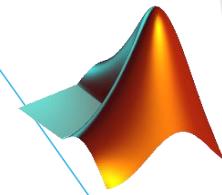


```
Command Window
>> feval('fun1',pi)

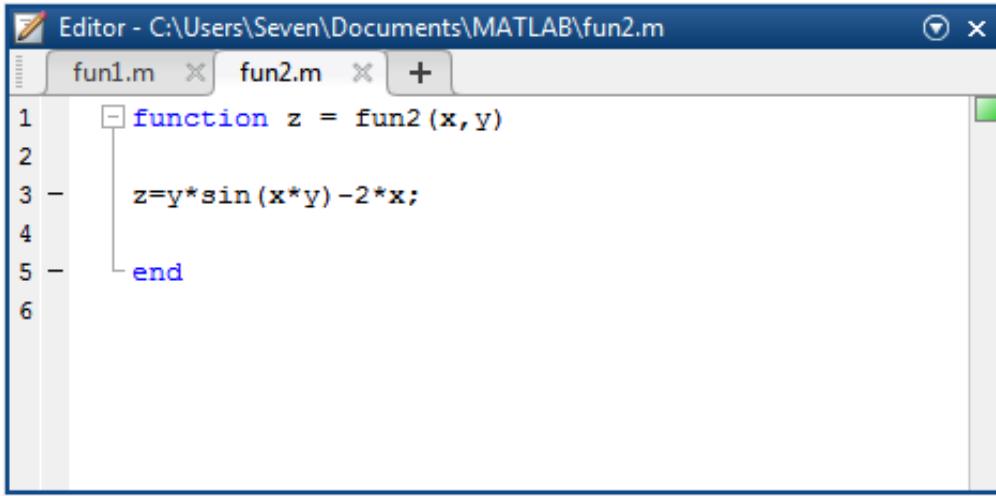
ans =
    1.0000

>> feval(@fun1,pi)

ans =
    1.0000
```

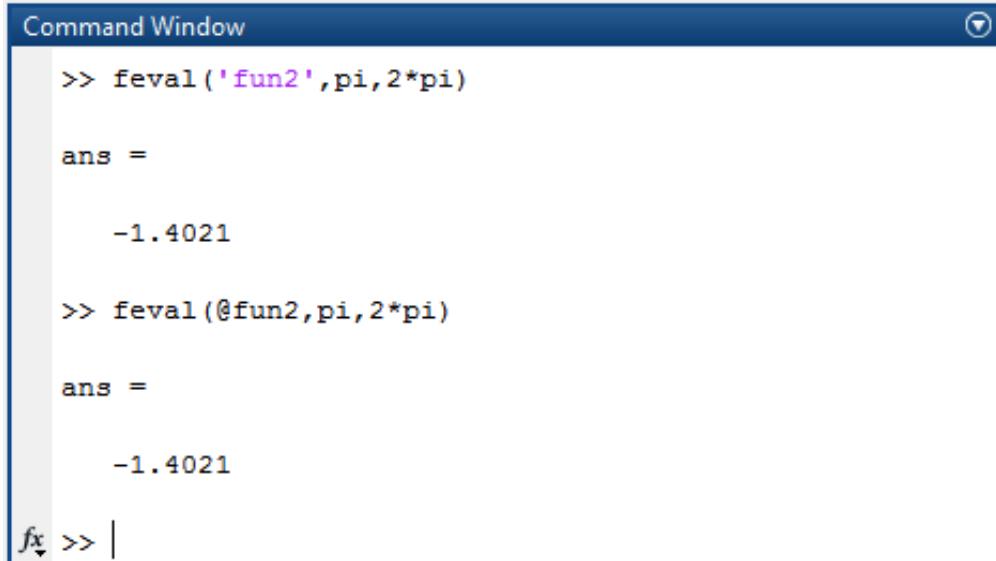


Ejemplo 2: Creamos la función **fun2.m** en dos variables.



```
Editor - C:\Users\Seven\Documents\MATLAB\fun2.m
fun1.m  fun2.m  +
1 function z = fun2(x,y)
2
3 z=y*sin(x*y)-2*x;
4
5 end
6
```

Ejecutamos en la ventana de comandos



```
Command Window
>> feval('fun2',pi,2*pi)

ans =

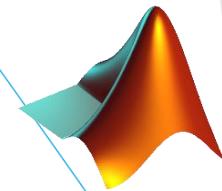
-1.4021

>> feval(@fun2,pi,2*pi)

ans =

-1.4021

fx >> |
```



Veamos otras formas de evaluación de una función.

Ejemplo 1: Utilice el comando **inline** para crear una función y luego ejecútelo.

Command Window

```
>> f=inline('3*x^2+sin(x)*cos(x)')

f =

    Inline function:
    f(x) = 3*x^2+sin(x)*cos(x)

>> f(pi) %evaluando f(x) en x=pi

ans =

    29.6088

>> g=inline('2*x+3*y-x*y-5','x','y')

g =

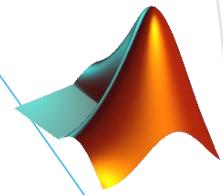
    Inline function:
    g(x,y) = 2*x+3*y-x*y-5

>> g(-3,5) %evaluando g(x,y) en x=-3 e y=5

ans =

    19

fx >>
```



Ejemplo 2: La evaluación de una función 'f' (m-archivo) creada en el editor de texto.

Las funciones **fun1** y **fun2** definidas en el editor.

```
Editor - C:\Users\Seven\Documents\MATLAB\fun1.m
fun1.m  x  fun2.m  x  +
1      [-] function y = fun1(x)
2
3      [-]     y=2*sin (x)-3*cos (x);
4
5      [-] end
6

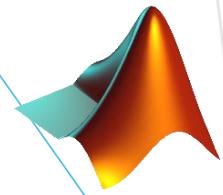
Editor - C:\Users\Seven\Documents\MATLAB\fun2.m
fun1.m  x  fun2.m  x  +
1      [-] function z = fun2(x,y)
2
3      [-]     z=y*sin (x*y)-2*x;
4
5      [-] end
6
```

Ejecución en la ventana de comandos de ambas funciones.

```
Command Window
>> fun1(pi) %evaluando f(x) en x=pi
ans =
    3.0000

>> fun2(pi,2*pi) %evaluando g(x,y) en x=pi e y=2*pi
ans =
   -1.4021

fx >> |
```



Problemas resueltos

Problema 1: Escriba y ejecute la función **distancia(xa,ya,xb,yb)** que calcula la distancia de los puntos **(xa,ya)** y **(xb,yb)**.

Solución: Programa en Matlab y ejecución del programa

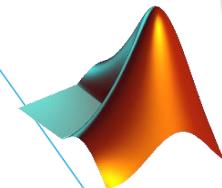
The screenshot shows the MATLAB R2015a interface. The top menu bar includes H..., PL..., A..., ED..., PU..., VI..., and a search bar. Below the menu is a toolbar with icons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Breakpoints, Run, and others. The left side features a 'Current Folder' browser showing files like distancia.m, fun1.m, etc., and a 'Command Window' at the bottom.

Editor - C:\Users\Seven\Documents\MATLAB\distancia.m

```
function d = distancia( xa,ya,xb,yb )
%xa: abscisa del punto A.
%ya: ordenada del punto A.
%xb: abscisa del punto B.
%yb: ordenada del punto B.
%resultado: distancia entre los 2 puntos A y B.
d=sqrt( (xb-xa) .^2+(yb-ya) .^2 );
end
```

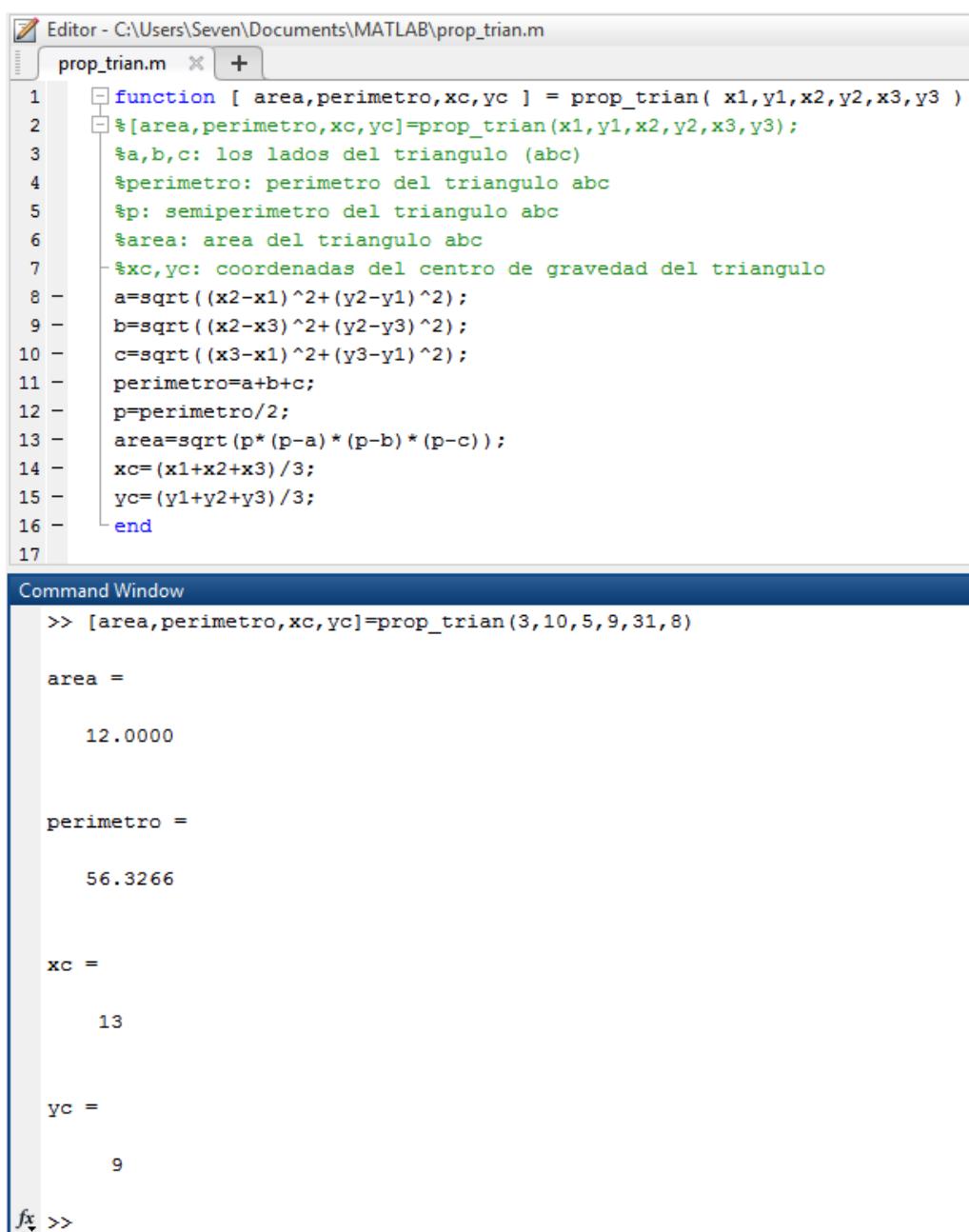
Command Window

```
>> d=distancia(1,1,2,2)
d =
1.4142
fx >>
```



Problema 2: Escribir una función **prop_trian(x1,y1,x2,y2,x3,y3)**; en Matlab que evalúe el área, perímetro y coordenadas del centro de gravedad de un triángulo cuyos vértices son **1,2,3**. La función creada debe de dar como resultado **[área,perímetro,xc,yc]**.

Solución: Programa en Matlab y ejecución del programa



```

Editor - C:\Users\Seven\Documents\MATLAB\prop_trian.m
prop_trian.m + 

1  [-] function [ area,perimetro,xc,yc ] = prop_trian( x1,y1,x2,y2,x3,y3 )
2  [-] %[area,perimetro,xc,yc]=prop_trian(x1,y1,x2,y2,x3,y3);
3  [-] %a,b,c: los lados del triangulo (abc)
4  [-] %perimetro: perimetro del triangulo abc
5  [-] %p: semiperimetro del triangulo abc
6  [-] %area: area del triangulo abc
7  [-] %xc,yc: coordenadas del centro de gravedad del triangulo
8  [-] a=sqrt((x2-x1)^2+(y2-y1)^2);
9  [-] b=sqrt((x2-x3)^2+(y2-y3)^2);
10 [-] c=sqrt((x3-x1)^2+(y3-y1)^2);
11 [-] perimetro=a+b+c;
12 [-] p=perimetro/2;
13 [-] area=sqrt(p*(p-a)*(p-b)*(p-c));
14 [-] xc=(x1+x2+x3)/3;
15 [-] yc=(y1+y2+y3)/3;
16 [-] end
17

Command Window
>> [area,perimetro,xc,yc]=prop_trian(3,10,5,9,31,8)

area =
12.0000

perimetro =
56.3266

xc =
13

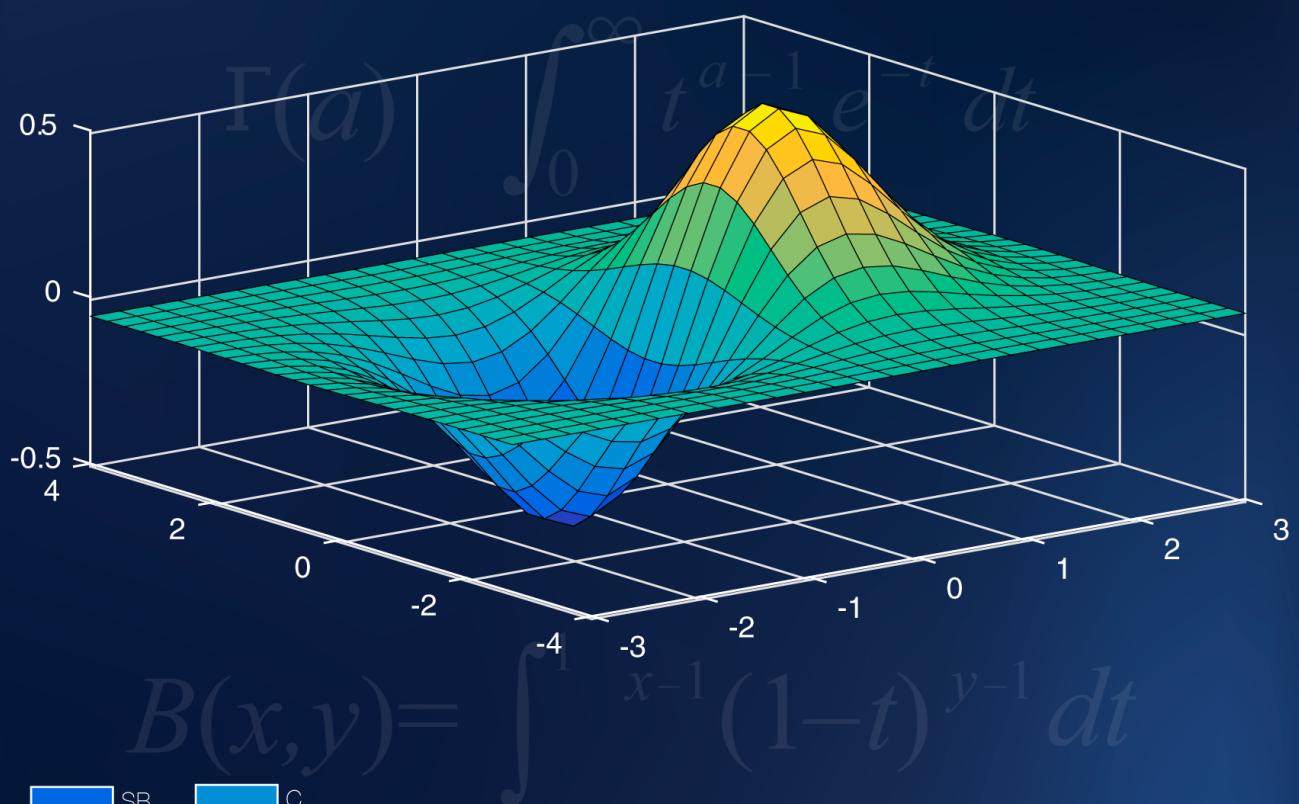
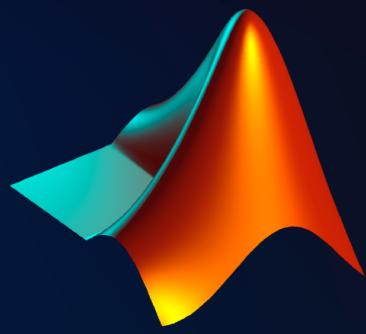
yc =
9

fx >>

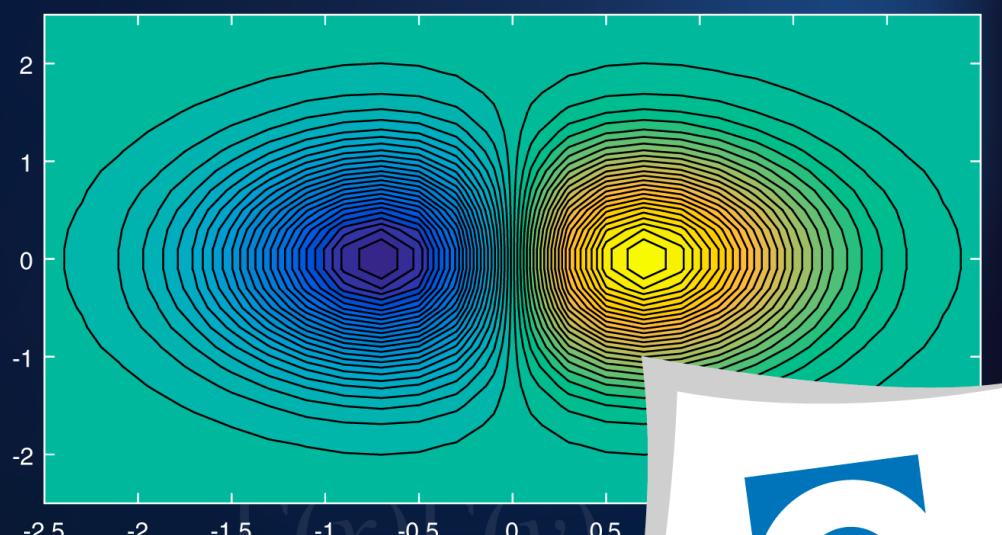
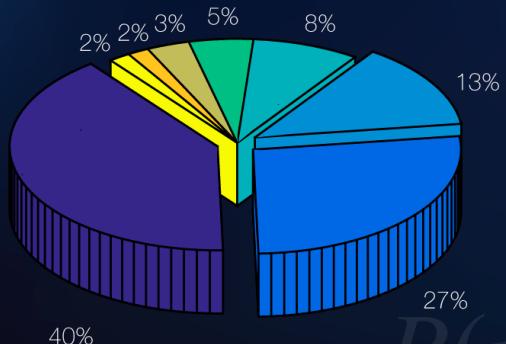
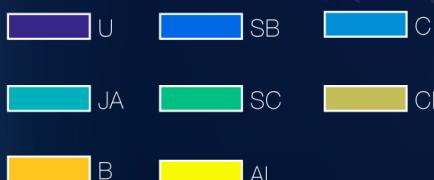
```

Curso MATLAB

Intermedio

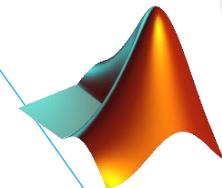


$$B(x,y) = \int_{-1}^1 x^{-1} (1-t)^{y-1} dt$$



Contenido

Bucles for y while.....	3
Sentencia if elseif else end	5
break y continue.....	8
switch y case	10
return	12



Sesión 6: SENTENCIAS DE DECISIÓN Y BUCLES FINITOS

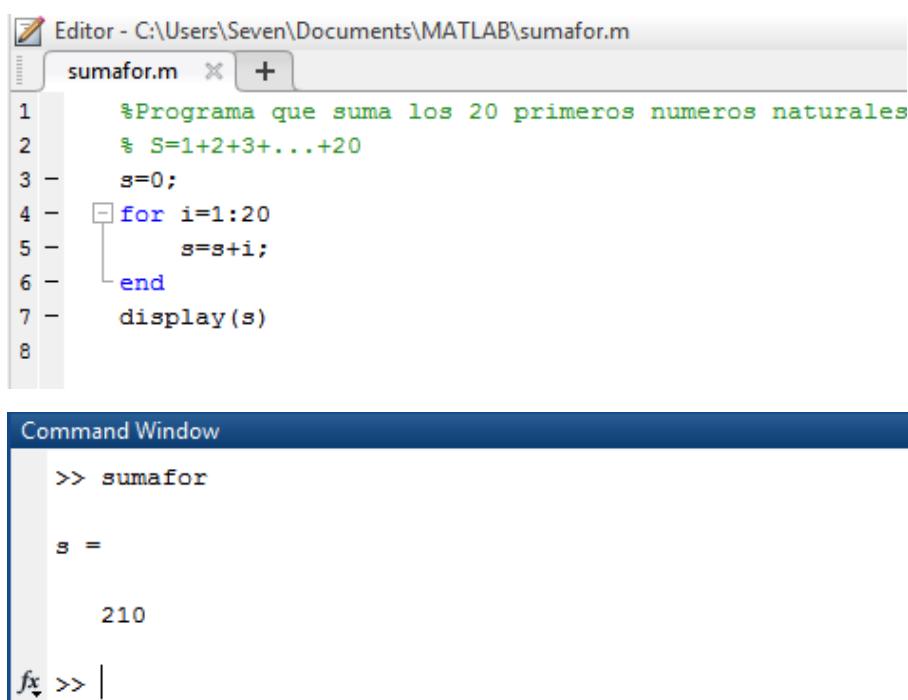
Bucles for y while

Los bucles permiten repetir un número determinado de veces un conjunto de sentencia. Entre ellos tenemos el bucle **for** y el bucle **while**.

- **Bucle for:** Su sintaxis es,

```
for vector=inicio:incremento:final
    sentencias que se deben ejecutar
end
```

Ejemplo: Hacer un programa para sumar los 20 primeros números naturales usando **for**.

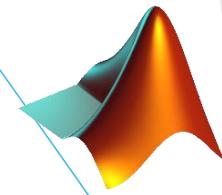


The screenshot shows the MATLAB environment. The Editor window displays a script named 'sumafor.m' with the following code:

```
1 %Programa que suma los 20 primeros numeros naturales
2 % S=1+2+3+...+20
3 s=0;
4 for i=1:20
5     s=s+i;
6 end
7 display(s)
8
```

The Command Window below shows the execution of the script:

```
>> sumafor
s =
210
>>
```



- **Bucle while:** Su sintaxis es,

while (condición)

sentencias (sentencias que se ejecutan cuando la **condición** es verdadera)

end

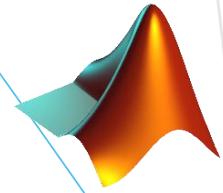
Ejemplo: Hacer un programa para sumar los 20 primeros números naturales usando **while**.

The screenshot shows the MATLAB interface with two windows open. The top window is the 'Editor' showing the script file 'sumawhile.m' with the following code:

```
1 %Programa que suma los 20 primeros numeros naturales
2 % S=1+2+3+...+20
3 s=0; i=1;
4 while i<=20
5     s=s+i;
6     i=i+1;
7 end
8 display(s)
9
```

The bottom window is the 'Command Window' showing the output of running the script:

```
>> sumawhile
s =
210
fx >> |
```



Sentencia if elseif else end

Mediante esta estructura, se pueden ejecutar secuencias de comandos si se cumplen determinadas condiciones. Su sintaxis es de la forma

```
if condición  
    comandos (Se ejecutan cuando la condición es cierta)  
end
```

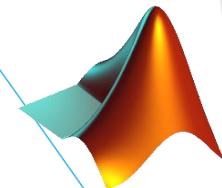
En este caso se ejecutan los comandos si la condición es cierta. Pero la sintaxis de este bucle puede ser más general.

```
if condición  
    comandos1 (se ejecutan cuando la condición es cierta)  
else  
    comandos2 (se ejecutan cuando la condición no se cumple)  
end
```

En este caso se ejecutan los comandos1 si la condición es cierta, y se ejecutan los comandos2 si la condición es falsa.

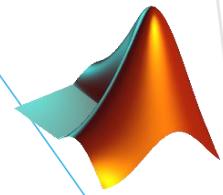
Las sentencias **IF**, al igual que las sentencias **FOR**, pueden ser anidadas. Cuando se anidan varias sentencias **IF** se utiliza la sentencia **ELSEIF**, cuya sintaxis general es la siguiente:

```
if condición1  
    comandos1  
elseif condición2  
    comandos2  
elseif condición3  
    comandos3  
...  
else  
end
```



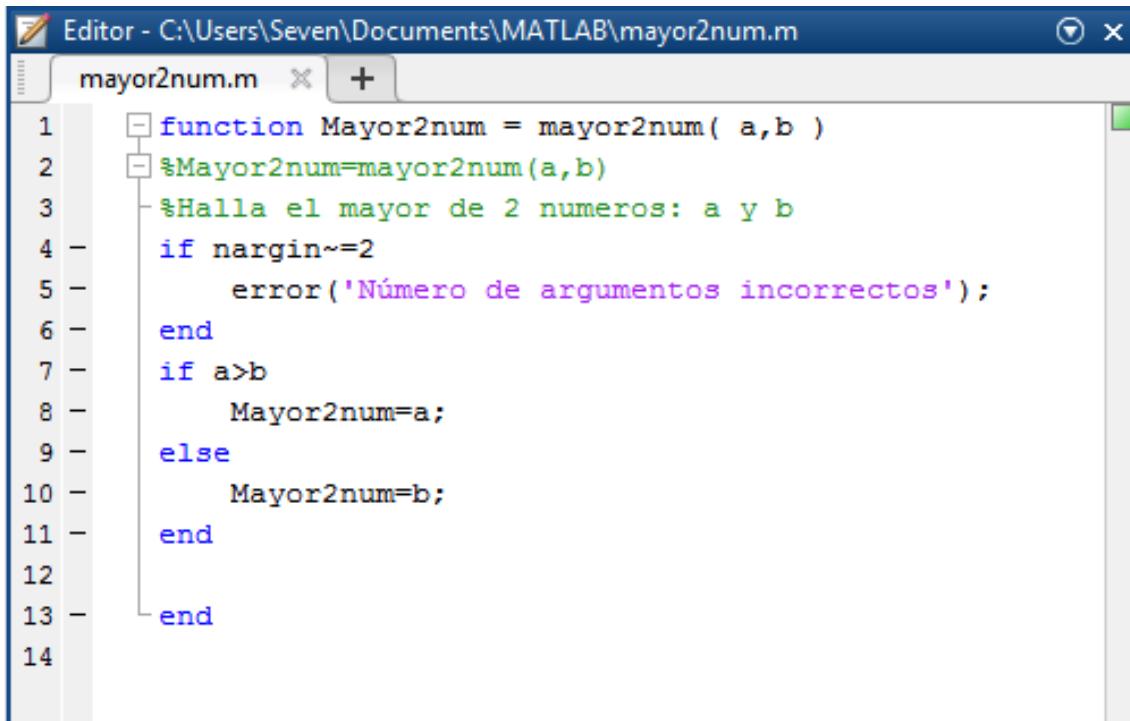
En este caso se ejecutan los comandos1 si condición1 es cierta, se ejecutan los comandos2 si condición1 es falsa y condición2 es cierta, se ejecutan los comandos3 si condición1 y condición2 son falsas y condición3 es cierta, y así sucesivamente. La sintaxis anidada anterior es equivalente, pero más rápida de ejecución, a la sintaxis sin anidar siguiente:

```
if condicion1
    comandos1 (Se ejecutan cuando la condición1 es cierta)
else
    if condición2
        comando2 (Se ejecutan cuando la condición1 no se cumple y sí la condición2).
    else
        if condición3
            comando3 (Se ejecutan cuando la condición1, condición2 no se cumplen y se cumple la condición3)
        else
            comandos4 (Se ejecutan cuando la condición1, condición2 y la condición3 no se cumplen)
        end
    end
    ...
end
```



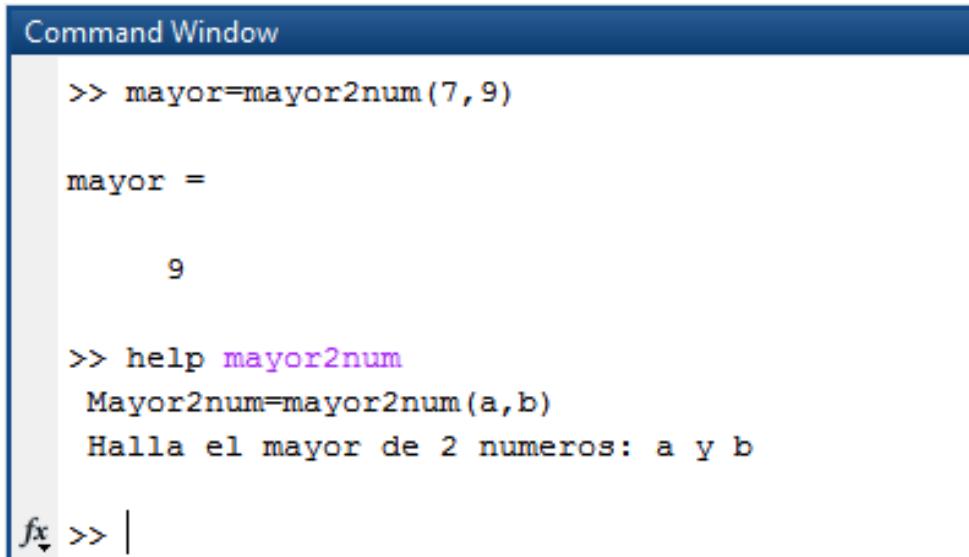
Ejemplo: Hacer un programa que calcule el mayor de 2 números a y b.

Código en el editor,



```
Editor - C:\Users\Seven\Documents\MATLAB\mayor2num.m
mayor2num.m + 
1 function Mayor2num = mayor2num( a,b )
2 %Mayor2num=mayor2num(a,b)
3 %Halla el mayor de 2 numeros: a y b
4 if nargin~=2
5     error('Número de argumentos incorrectos');
6 end
7 if a>b
8     Mayor2num=a;
9 else
10    Mayor2num=b;
11 end
12
13 end
14
```

Ejecución del programa en la ventana de comandos,

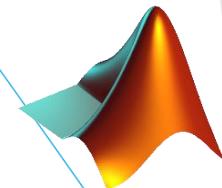


```
Command Window
>> mayor=mayor2num(7,9)

mayor =
9

>> help mayor2num
Mayor2num=mayor2num(a,b)
Halla el mayor de 2 numeros: a y b

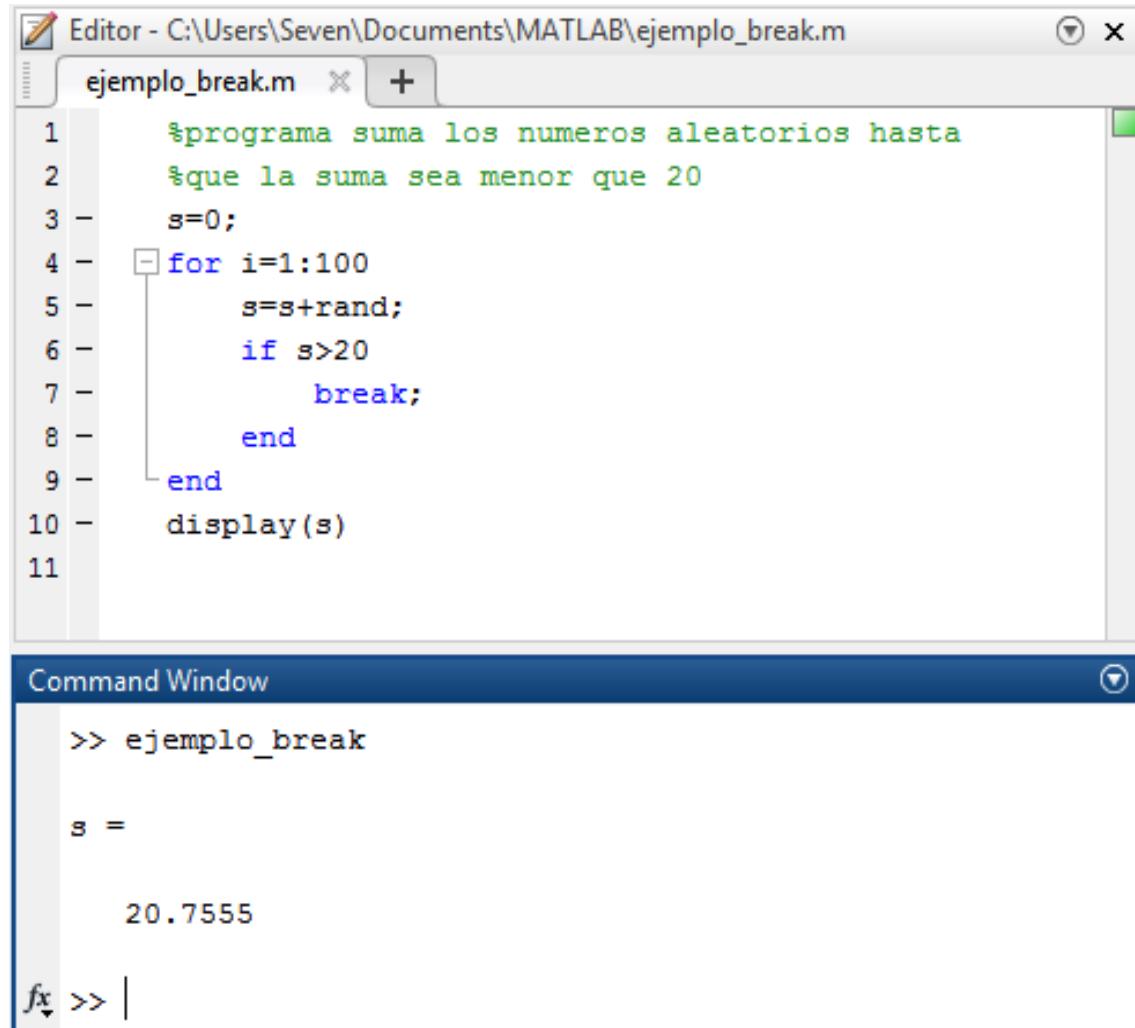
fx >> |
```



break y continue

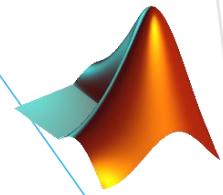
break: La sentencia **break** finaliza la ejecución de un bucle **for** o **while** en el cual se encuentra, luego el programa sigue la ejecución en la siguiente sentencia fuera del bucle.

Ejemplo de sentencia break



The screenshot shows the MATLAB environment with two windows open:

- Editor - C:\Users\Seven\Documents\MATLAB\ejemplo_break.m**: This window displays the MATLAB script `ejemplo_break.m`. The code initializes a sum `s=0`, enters a `for` loop from 1 to 100, adds a random number to `s` each iteration, checks if `s` is greater than 20, and if so, uses the `break` statement to exit the loop. Finally, it displays the value of `s`.
- Command Window**: This window shows the execution of the script. The user types `>> ejemplo_break` and presses Enter. The output shows the variable `s` assigned the value `20.7555`.



continue: La sentencia **continue** pasa el control a la iteración siguiente en el bucle **for** o **while** en el cual aparece ignorando las restantes sentencias en el cuerpo del bucle.

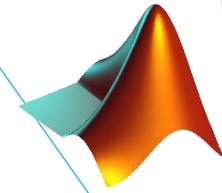
Ejemplo de sentencia continue

The screenshot shows the MATLAB interface with two windows open. The top window is the 'Editor' showing the script file 'ejemplo_continue.m' with the following code:

```
%Programa que imprime los numeros desde
%1 hasta 10 excepto los multiple de 3
for i=1:10
    if (rem(i,3)==0)
        continue;
    end
    disp(i)
end
```

The bottom window is the 'Command Window' showing the output of running the script:

```
>> ejemplo_continue
1
2
4
5
7
8
10
```



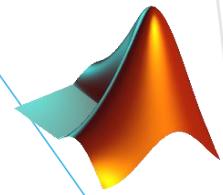
switch y case

La sentencia **switch** realiza una función análoga a un conjunto de **if...elseif** concatenados.

Al principio se evalúa la **switch_expresion**, cuyo resultado debe ser un número escalar o una cadena de caracteres. Este resultado se compara con las **case_expr**, y se ejecuta el bloque de sentencias que corresponda con ese resultado. Si ninguno es igual a **switch_expresion** se ejecutan las sentencias correspondientes a **otherwise**.

Su sintaxis es de la forma

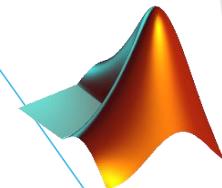
```
switch switch_expresion
    case case_expr1,
        bloque1
    case {case_expr2, case_expr3, case_expr4,...}
        bloque2
    ...
    otherwise % opción por defecto
        bloque3
end
```



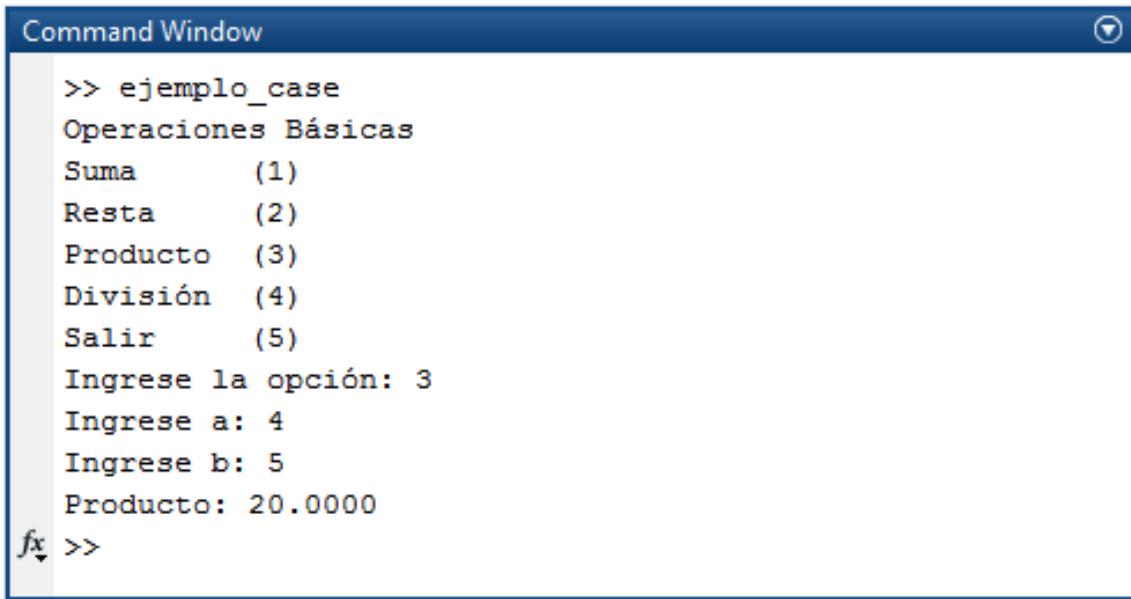
Ejemplo de sentencia switch

Editor - C:\Users\Seven\Documents\MATLAB\ejemplo_case.m

```
1 %Programa con multiples opciones o casos
2 disp('Operaciones Básicas')
3 disp('Suma      (1)')
4 disp('Resta     (2)')
5 disp('Producto  (3)')
6 disp('División  (4)')
7 disp('Salir     (5)')
8 n=input('Ingrese la opción: ');
9 switch(n)
10 case 1
11     a=input('Ingrese a: ');
12     b=input('Ingrese b: ');
13     fprintf('Suma: %6.4f\n',a+b);
14 case 2
15     a=input('Ingrese a: ');
16     b=input('Ingrese b: ');
17     fprintf('Resta: %6.4f\n',a-b);
18 case 3
19     a=input('Ingrese a: ');
20     b=input('Ingrese b: ');
21     fprintf('Producto: %6.4f\n',a*b);
22 case 4
23     a=input('Ingrese a: ');
24     b=input('Ingrese b: ');
25     fprintf('suma: %6.4f\n',a/b);
26 case 5
27     disp('Programa finalizado ... Gracias');
28 otherwise
29     disp('Opcion no valida');
30 end
```



Ejecución en la ventana de comandos:

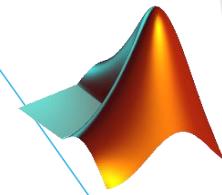


```
Command Window
>> ejemplo_case
Operaciones Básicas
Suma      (1)
Resta     (2)
Producto  (3)
División (4)
Salir     (5)
Ingrese la opción: 3
Ingrese a: 4
Ingrese b: 5
Producto: 20.0000
fx >>
```

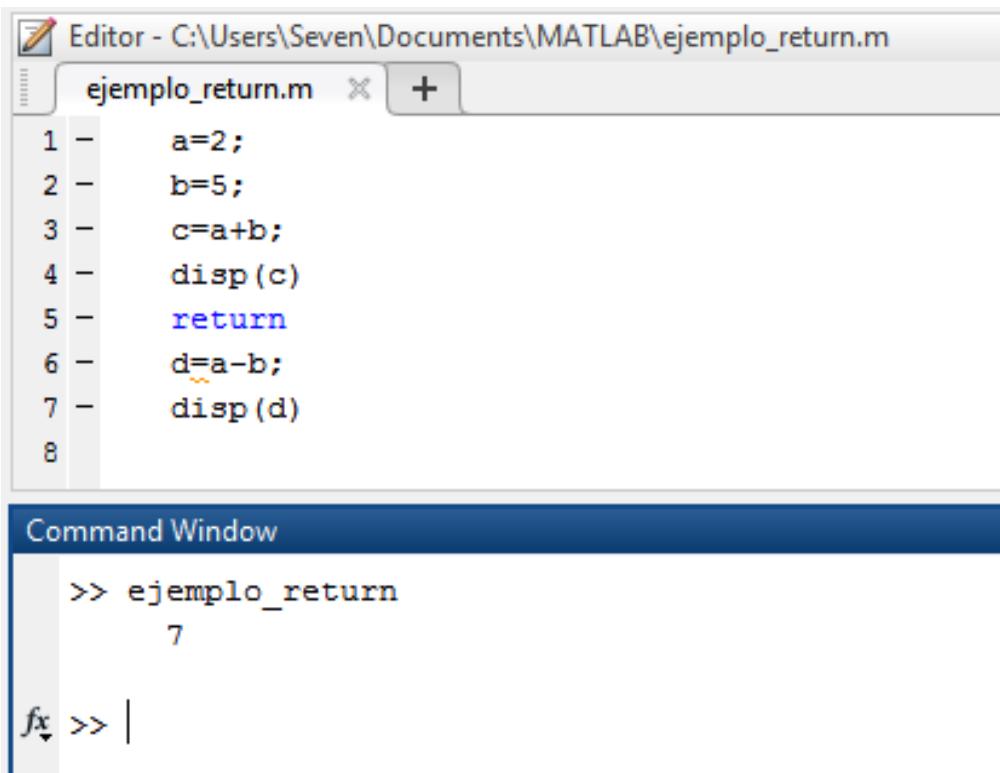
return

return fuerza a MATLAB a devolver el control a la función invocada antes de que alcance el final de la función. La función invocada es la función que llama al script o función que contiene el llamado a **return**. Si se llama a la función o el script que contiene **return** directamente, no hay ninguna función invocada y MATLAB devuelve el control al **prompt** en la línea de comandos.

Nota: Tenga cuidado cuando utilice el **return** dentro de los bloques condicionales, como **if** o **switch**, o dentro del bucle sentencias de control, como **for** o **while**. Cuando MATLAB alcanza una sentencia **return**, este no sólo sale del bucle; que sale de la secuencia de comandos o función y devuelve el control a la función de invocación o símbolo del sistema.



Ejemplo:



The screenshot shows the MATLAB environment. The top part is the 'Editor' window titled 'Editor - C:\Users\Seven\Documents\MATLAB\ejemplo_return.m'. It contains the following MATLAB script:

```
1 - a=2;
2 - b=5;
3 - c=a+b;
4 - disp(c)
5 - return
6 - d=a-b;
7 - disp(d)
8
```

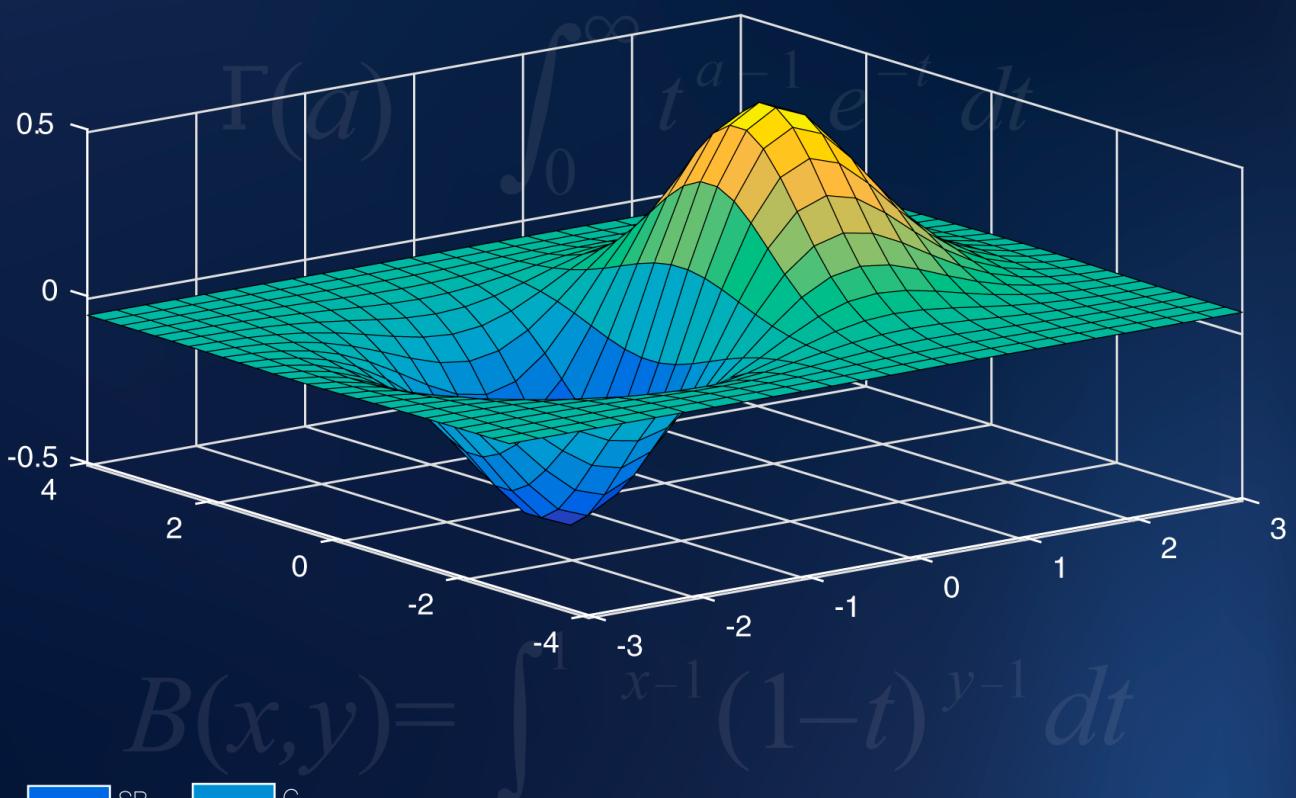
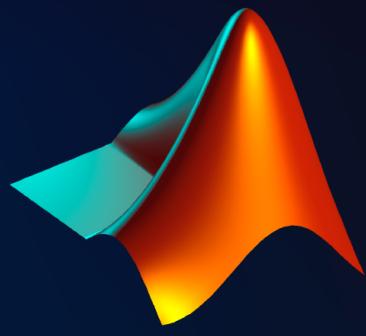
The bottom part is the 'Command Window' titled 'Command Window'. It shows the output of running the script:

```
>> ejemplo_return
    7
```

The command window also has a cursor at the end of the command line.

Curso MATLAB

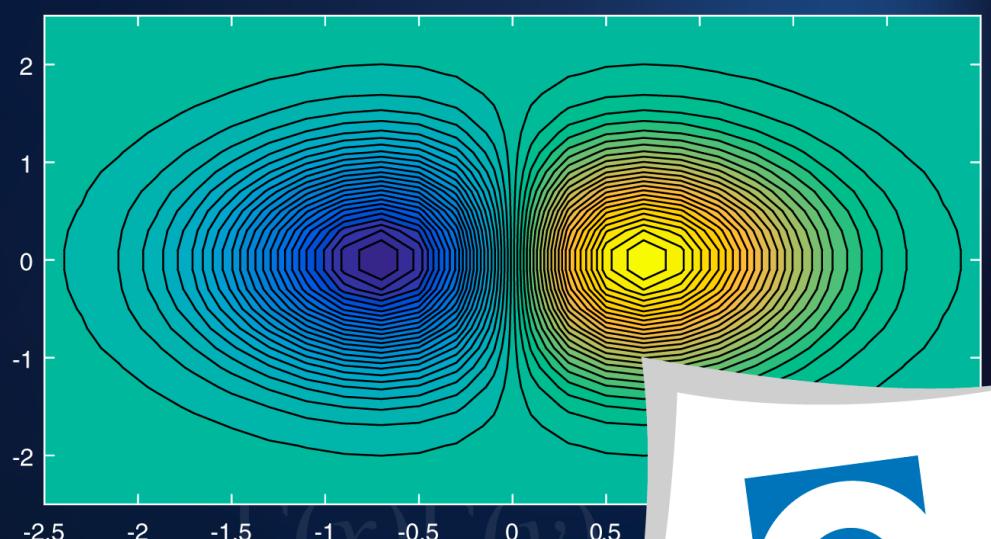
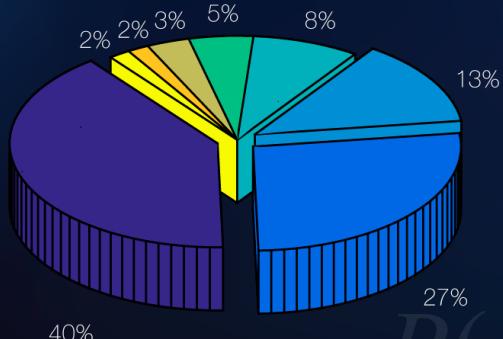
Intermedio

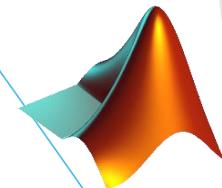


U SB C

JA SC CNI

B AL





Matemática Computacional y Aplicada

MATLAB Intermedio

Examen Final

Nota: Resolver los ejercicios haciendo uso de los comandos y herramientas aprendidas en las sesiones.

- i. **Cada ejercicio debe estar resuelto en un archivo .m ya sea un programa o una función incluido comentarios de ayuda y mensajes de error.**
 - ii. **Los gráficos deben tener sus respectivas etiquetas de coordenadas, leyendas y título, además de mostrarse en una misma ventana para cada pregunta. También adjuntar las capturas correspondientes.**
1. El punto dado en coordenadas cilíndricas (2.5,0.8,2).
 - a. Graficar el punto con la marca de un asterisco de color azul, graficar como vector el punto (línea que une el punto con el origen).
 - b. Expresar el punto en coordenadas esféricas.
 2. Identificar la superficie dada pasándola a coordenadas cartesianas y realizar los gráficos de mallas y su contorno.
 - a. $z = r^2$
 - b. $r^2 + z^2 = 36$, no usar el comando *sphere*; y realizar el gráfico centrado en la coordenada (1,2,3).
 3. Resolver:
 - a. Hallar los 100 primeros números primos después del número 100 (no usar el comando *isprime*) y calcular la suma de todos ellos (sin usar el comando *sum*).
 - b. Generar 20 números enteros de forma aleatoria y ordenarlos de forma ascendente (Sin hacer uso del comando *sort*).
 4. Realizar una función con nombre ***sistemaLineal*** con dos variables de entrada que corresponden a la matriz de coeficientes y a la matriz de términos independientes. Realizar el programa de tal forma que:

- a. Si los datos ingresados no son correctos (ejemplo: tipo de dato incorrecto, dimensiones diferentes, etc.) dar un mensaje de error conveniente.
 - b. Si los datos ingresados son correctos, verificar si el sistema es compatible o incompatible y mostrar el mensaje en el caso que sea incompatible.
 - c. Si es compatible resolver si es determinado o indeterminado, en caso fuera indeterminado mostrar el mensaje.
 - d. Si el sistema de ecuaciones lineales es compatible determinado hallar la solución.
 - e. Observación: Asigne la variable o variables de salida correspondientes.
5. Los siguientes datos corresponden al peso en Kg de un conjunto de personas.
- | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|
| 66 | 63 | 62 | 66 | 70 | 69 | 59 | 68 | 71 | 67 | 58 |
| 79 | 73 | 64 | 72 | 64 | 67 | 67 | 61 | 76 | 63 | 62 |
| 61 | 76 | 68 | 71 | 67 | 69 | 68 | 67 | 62 | 57 | 73 |
| 67 | 61 | 69 | 65 | 65 | 54 | 67 | 74 | 65 | 56 | 83 |
| 72 | 68 | 69 | 64 | 62 | 81 | 71 | 64 | 75 | 78 | 67 |
| 70 | 66 | 59 | 80 | 69 | 63 | 61 | 74 | 67 | 66 | 58 |
| 71 | 69 | 65 | 75 | 52 | 66 | 70 | 57 | 68 | 66 | 70 |
| 77 | 66 | 60 | | | | | | | | |
- a. Ingresar los datos como vector o matriz y obtener una distribución de datos en intervalos de amplitud 5, mostrar la distribución y las frecuencias absolutas.
 - b. Calcular las principales medidas de tendencia central y mostrarlos junto a un mensaje de interpretación.
 - c. Realizar el gráfico estadístico de las frecuencias absolutas con el comando que usted elija para visualizar mejor el resultado.
 - d. Hallar el porcentaje de personas con peso menor o igual a 65 Kg y mayor o igual a 80 Kg. Realizar el gráfico de sectores y desplazar hacia afuera los sectores de personas con peso menor o igual a 65 Kg y mayor o igual de 80 Kg.