MATLAB

Índice

Bloque II: TUTORIAL DEL TOOLBOX DE MATEMÁTICA SIMBÓLICA

Bloque III: TUTORIAL DEL TOOLBOX DE SEÑALES Y SISTEMAS

PRÓLOGO

MATLAB es el primer **entorno de cálculo técnico** hoy en día. La primera versión fue escrita en la Universidad e New México y en la Universidad de Stanford a finales de los años 70, y tenía como finalidad utilizarse en cursos de teoría de matrices, álgebra lineal y análisis numérico.

Hoy en día, las capacidades de MATLAB se extienden mucho más allá del original "laboratorio matricial". MATLAB es un sistema interactivo y un **lenguaje de programación** para científicos en general y para cálculos técnicos. Su elemento de datos básico es una matriz que no requiere dimensionamiento. Esto permite la solución de muchos problemas numéricos en una pequeña fracción del tiempo que llevaría escribir un programa en un lenguaje tal como FORTRAN, BASIC o C. Más aún, las soluciones del problema se expresan en MATLAB casi exactamente a como se escriben matemáticamente.

Las matemáticas es el lenguaje común de gran parte de la ciencia y de la ingeniería. Matrices, ecuaciones diferenciales, arrays de datos y gráficas son un bloque de construcción básicos de las matemáticas aplicadas y de MATLAB. Es la base matemática fundamental que hace que MATLAB sea accesible y potente.

Algunos ejemplos de su uso podrían ser:

-Una estudiante graduada en Físicas analiza datos de sus experimentos con campos magnéticos de superconductores.
-Un restaurador de parques de atracciones conocido internacionalmente lo utiliza para modelizar los sistemas de control de sus fuentes de agua.
-Una gran compañía de alimentación analiza cómo el horno de microondas cocina pizzas.
-Una compañía de televisión por cable investiga esquemas de codificación y compresión para la TV digital.
-Un fabricante de equipamiento deportivo modela los golpes de golf.
-Un estudiante de primaria a prende las tablas de multiplicar.

En todos estos casos, y en miles más, el fundamento matemático de MATLAB resultó útil en lugares y aplicaciones mucho más allá de las que contemplamos originalmente.

LIMITACIONES DE LA VERSIÓN DEL ESTUDIANTE.

-No necesita (aunque se recomienda) coprocesador matemático.
-La capacidad del tamaño de las matrices está limitada a 8192 elementos, con el número de filas o columnas restringido a 32.
-Sólo imprime en Windows, Macintosh y en dispositivos PostScript.
-Sólo está disponible en licencias de un único usuario (no soporta red).
-No puede enlazar dinámicamente subrutinas en C o en FROTRAN (archivos .MEX).
-Los únicos *toolboxes* que se pueden utilizar con ella son los que van incluidos en el *toolbox de matemática simbólica* y en el *toolbox de señales y sistemas*.
-Proporciona un descuento de actualización de un usuario estudiante que desee comprar la versión profesional.

TOOLBOXES

MATLAB es al mismo tiempo un entorno y un lenguaje de programación. Uno de sus puntos fuertes es el hecho de que el lenguaje MATLAB le permite construir sus propias herramientas reutilizables. Puede crear fácilmente sus propias funciones y programas especiales (conocidos como archivos .M) en código MATLAB. Cuando escriba más y más funciones MATLAB para tratar ciertos problemas, podría estar tentado a agrupar juntas, por conveniencia suya, funciones relacionadas en directorios especiales. Esto conduce directamente al concepto de *toolbox*: una colección especializada de archivos .M para trabajar clases particulares de problemas.

Los *toolboxes* son algo más que simples colecciones de funciones útiles: representan los esfuerzos de investigadores líderes a nivel mundial en campos tales como control, procesamiento de señales, e identificación de sistemas. Debido a esto, los *toolboxes* de aplicación de MATLAB le permiten "descansar en los hombros" de científicos de renombre a nivel mundial.

Todos los *toolboxes* se construyen directamente por encima de MATLAB. Esto tiene algunas implicaciones muy importantes para usted:

-Cada *toolbox* se diseña utilizando métodos numéricos robustos, precisión muy sólida y años de experiencia en MATLAB.

-Obtiene una integración sin fisuras e inmediata con SIMULINK y con cualesquiera otros *toolboxes* que pueda poseer.

-Como todos los *toolboxes* están escritos en código MATLAB, puede beneficiarse del enfoque de sistema abierto de MATLAB. También puede examinar los archivos .M, modificarlos, o utilizarlos como plantillas cuando esté creando sus propias funciones.

-Cada *toolbox* está disponible en cualquier plataforma que funcione con MATLAB.

Presentamos a continuación una lista de *toolboxes* profesionales actualmente disponibles de The Math Works. Esta lista no es, en absoluto, estática –muchos están siendo creados cada año-.

Control System Toolbox. Base de la familia de *toolboxes* de diseño de sistemas de control.

Frecuency-Domain System Identification Toolbox. Es un conjunto de archivos .M para modelar sistemas lineales basados en medidas de la respuesta en frecuencia del sistema.

Fuzzy Logic Toolbox. Lógica borrosa.

Higher-Order Spectral Analisys Toolbox. Procesamiento de señales utilizando espectros de orden elevado.

Image Processing Toolbox. Herramientas para el procesamiento de imágenes y desarrollo de algoritmos.

Model Predictive Control Toolbox. Control predictivo basado en modelos.

Mu-Analisys and Sybthesis Toolbox. Herramientas especializadas para control óptimo H_∞.

NAG Foundation Toolbox. Más de 200 funciones de cálculo numérico de subrutinas FORTRAN.

Neural Network Toolbox. Diseño y simulación de redes neuronales.

Nonlinear Control Design Toolbox. Adición a SIMULINK para el diseño de sistemas de control no lineal.

Optimization Toolbox. Optimización de funciones generales y no lineales.

Quantitative Feedback Theory Toolbox. Diseño de controladores para sistemas inciertos.

Robust Control Toolbox. Análisis y diseño de sistemas de control robustos.

Signal Processing Toolbox. Herramientas para el procesamiento de señales.

SIMULINK. Es una extensión a MATLAB que añade un entorno gráfico para modelar, simular y analizar sistemas dinámicos lineales y no lineales.

SIMULINK Real-Time Workshop. Adición a SIMULINK para realización rápida de prototipos y simulación en tiempo real.

Splines Toolbox. Herramientas para construir Splines.

Statistics Toolbox. Análisis de datos estadísticos.

Symbolic Math Toolbox. Conjunto de herramientas, basadas en MAPLE V para cálculo simbólico y aritmética con precisión variable.

System Identification Toolbox. Herramientas para la estimación e identificación (encontrar un modelo matemático para un sistema físico basado solamente en un registro de entradas y salidas del sistema).

Los requisitos mínimos del sistema son: 386, 486 o Pentium, Microsoft Windows 3.1, disquetera de 3.5", monitor y ratón. 15Mb de espacio en el disco duro y 8Mb de memoria extendida.

El *cuaderno de notas* de MTALAB es una integración dinámica de MATLAB y Microsoft Word 6.0 que le permite construir documentos de procesamiento de texto interactivos que contienen órdenes y gráficos activos en MATLAB.

Utilizando el cuaderno de notas puede crear un *libro-M*, un documento estándar de Microsoft Word que contiene no solamente texto, sino también órdenes MATLAB y la salida de estas órdenes.

BLOQUE I
TUTORIAL DE MATLAB

I.1.-CARACTERÍSTICAS BÁSICAS.

Veamos en primer lugar algunas de las características más elementales de MATLAB, como es la sintaxis básica.

a)Se observa que al entrar en MATLAB, aparte de algunos mensajes, aparece el indicador (PROMPT)

EDU>>

que indica que MATLAB está esperando que usted le de alguna orden (en la versión profesional el PROMP es simplemente >>, que es el que usaremos nosotros para abreviar).

b)MATLAB puede ser utilizado como una calculadora corriente, esto es, podemos realizar la operación 25*3 tecleando esto mismo y pulsando finalmente la tecla INTRO.
teclear
>>25*3
se obtendrá la siguiente respuesta:
ans=
75
>>
que significa que la respuesta es 75 y que vuelve a estar a la espera de recibir más órdenes.

Realmente lo que hace MATLAB es guardar el resultado en una variable genérica del sistema llamada **ans** (abreviatura de answer: "respuesta").

Esta variable (y cualquier otra) podemos consultar su valor, simplemente tecleando su nombre:
Teclear
>>ans
se obtiene:
ans=
75
>>

Las operaciones matemáticas básicas entendidas por MATLAB son

Operación	Operador	Ejemplo
1-Suma: **a+b**	+	25+3
2-Resta: **a-b**	**-**	7-3
3-Multiplicación: **a x b**	*****	2.15*6
4-División: **a : b**	**/** o ****	4/5 = 5\4
5-Potencia: $\mathbf{a^b}$	**^**	4^3

El orden en que estas operaciones se evalúan en una expresión determinada viene dado por las reglas usuales de precedencia que se pueden resumir como sigue:

Las expresiones se evalúan de izquierda a derecha, con la operación de potencia teniendo el orden de precedencia más alto, seguida por multiplicación y división, que tienen ambas igual precedencia y seguidas, finalmente, por suma y resta que tienen ambas la misma precedencia.

Se pueden emplear paréntesis para alterar esta usual ordenación, de la forma por todos conocida.

Los números complejos se manejan de la forma habitual, guardando las letras **i** o **j**, para denominar la unidad imaginaria (Ö-1). (Pueden dejarse blancos entre parte real e imaginaria, pero para evitar problemas de confusión entre columnas, en matrices no debe dejarse esa separación, o, en su defecto, deberán usarse paréntesis adicionales).

c)**Espacio de trabajo de MATLAB**.

Cuando trabajamos en la ventana de orden, MATLAB recuerda las órdenes que introducimos, así como los valores de cualquier variable que creemos. Estas órdenes y variables se dice que *residen* en el *espacio de trabajo* de MATLAB, y pueden volver a llamarse siempre que se desee.

d)Variables.

Como cualquier otro lenguaje de computador, MATLAB tiene reglas acerca de los nombres de las variables, como son:

-No deben contener espacios en blanco ni caracteres especiales (excepto el guión de subrayado, que si se utiliza en lugar del espacio).
-Hay sensibilidad respecto a MAYÚSCULAS y minúsculas (las variables Mesa y mEsa son consideradas distintas por MATLAB).
-La longitud máxima distinguible es de hasta 19 caracteres.
-Siempre deben comenzar por una letra, seguida de cualquier número de letras, dígitos o guiones de subrayado).

MATLAB tiene algunas **variables especiales predefinidas**, como:

ans: Nombre por defecto de la variable usada para los resultados.
pi: Número pi (3.14159...)
eps: Número más pequeño tal que, cuando se le suma 1, crea un número en coma flotante en el computador mayor que 1 (normalmente 2.2204e-016).
inf: Infinito (p.e.: 1/0).
NaN: Magnitud no numérica (Not-a-Number) (p.e., 0/0).
i y **j**: Unidad imaginaria (i^2=-1).
realmin: El número real positivo más pequeño que es utilizable (2.2251e-308).
realmax: El número real positivo más grande que es utilizable (1.7977e+308).

Estos valores son los predefinidos. Si los cambiamos, habrá que reiniciar MATLAB para que vuelvan a tener los valores originales.

Podemos borrar variables mediante la orden **clear <variables>** (donde podemos poner varias variables separadas por blancos). Si no se indica ninguna variable se borrarán todas (ojo porque no pide confirmación).

Se puede preguntar en cualquier momento por las variables definidas mediante la orden **who**. (La orden **whos** nos da más información que la anterior, si es que lo necesitamos). Pero no nos da sus valores. Para esto bastará con teclear el nombre de la variable que deseemos visualizar.

e)Abreviar órdenes.

Se puede utilizar los cursores para recordar órdenes de líneas anteriores y corregir dentro de una misma línea.

f)Comentarios.

Cualquier cosa que se ponga detrás del símbolo **%** se ignorará a efectos de interpretación. Se recomienda el uso de estos comentarios para describir detalladamente los archivos .M.

g)Si por alguna causa no tuviésemos suficiente con una línea para darle una orden a MATLAB, pueden ponerse los tres puntos (**...**) al final de la misma, lo que indicará a MATLAB que la orden no ha acabado y continúa en la línea siguiente.

h)**coma** y **punto y coma**.

Los argumentos horizontales se pueden separar por **blancos** o por una coma (**,**). Esto hará que se observe el resultado de la orden en la línea siguiente. Si no se desea que un determinado resultado se visualice, bastará con poner detrás suyo el símbolo punto y coma (**;**). Además, este símbolo sirve para separar filas entre sí para un array (lo mismo que hace la tecla INTRO).

i)Pueden ponerse varias órdenes en una misma línea, usando cualquiera de esos separadores (coma o punto y coma).

j)Puede interrumpirse MATLAB en cualquier momento, pulsando simultáneamente las teclas **Ctrl.-C**.

k)También podemos terminar la ejecución de MATLAB escribiendo la orden **quit**.

l)Almacenar y recuperar datos.

Podemos usar **Save Workspace as ...** del menú **File** para guardar las variables actuales. Análogamente, la opción **Load Workspace...** en el menú **File** permite cargar variables de un espacio de trabajo guardado previamente (ojo al cargar, porque si existen variables con el mismo nombre, sus valores serán sustituidos por los cargados).

Existen además las órdenes **load** y **save** que podemos usar con más detalle.

m)Ayuda en línea.

MATLAB dispone de tres formas de prestar ayuda en línea, mediante las órdenes **help, lookfor, help** del menú.

m1) La orden **help**.
Sin argumentos proporciona una ayuda genérica.
Con argumento: **help <orden>** nos da información de la orden pedida.
m2) La orden **lookfor**.
Proporciona ayuda buscando a través de todas las primeras líneas de las ayudas a temas de MATLAB, devolviendo aquellos que contienen una palabra clave que hay que especificar. Lo más importante es que dicha palabra no necesita ser una orden (en el apartado de ficheros .M describiremos con más detalle esta interesante opción).

I.2.-FORMATO DE VISUALIZACIÓN DE NÚMEROS

Por defecto, si un resultado es entero, MATLAB lo visualiza como entero. Si es real lo visualiza con cuatro dígitos a la derecha del punto decimal. Si los dígitos significativos en el resultado están fuera de este rango, se utiliza la notación científica habitual. En cualquier caso, puede inhibirse o modificarse este comportamiento especificando un formato numérico diferente, mediante la opción **Numerical Format** en el menú **Options** o escribiendo la orden apropiada en MATLAB.

Las posibilidades son las siguientes:

Format rat (racional) 37/3
Format short (corto) 12.3333
Format long (largo) 12.33333333333334
Format hex (hexadecimal) 4028aaaaaaaaaab
Format bank (bancario) 12.33
Format plus (signo) +
Format Short e (cientif.) 1.2333e+001
Format Long e (cientif.) 1.233333333333334e+001

Hay que reseñar que el formato de representación interno de MATLAB no cambia, solamente cambia su presentación en pantalla.

I.3.-FUNCIONES MATEMÁTICAS COMUNES.

A continuación damos una lista de las funciones matemáticas más comúnmente usadas y su representación en MATLAB.

Función	Acción	Ejemplo
abs(x)	Valor absoluto o módulo de un complejo	abs(-2.7)= 2.7 abs(3+4j)= 5

acos(x)	Arco coseno (en radianes)	
acosh(x)	Arco coseno hiperbólico	
angle(z)	Argumento (rd)	
asin(x)	Arco seno (en radianes)	
asinh(x)	Arco seno hiperbólico	
atan(x)	Arco tangente (en radianes)	
atan2(x,y)	Arco tangente en los cuatro cuadrantes	
atanh(x)	Arco tangente hiperbólico	
ceil(x)	Redondea hacia +infinito	ceil(-3.7)= -3
conj(x)	Complejo conjugado	conj(-2-3j)=-2+3i
cos(x)	Coseno (en radianes)	
cosh(x)	Coseno hiperbólico	
exp(x)	Exponencial	
fix(x)	Redondea hacia cero	fix(-3.7)= -3
floor(x)	Redondea hacia –infinito	floor(-3.7)= -4
imag(x)	Parte imaginaria de un complejo	imag(-2-3j)=-3
log(x)	Logaritmo neperiano	
log10(x)	Logaritmo decimal	
real(x)	Parte real de un complejo	
rem(x,y)	Resto de la división x/y (módulo)	rem(5,3)= 2
round(x)	Redondea hacia el entero más próximo	round(-3.7)=-4
sign(x)	Signo del argumento (-1, +1 o 0)	sign(-3.7)= -1
sin(x)	Seno (en radianes)	
sinh(x)	Seno hiperbólico	
sqrt(x)	Raíz cuadrada	sqrt(9)= 3
tan(x)	Tangente (en radianes)	
tanh(x)	Tangente hiperbólica	

I.4.-ARRAYS (MATRICES)

Se ponen entre corchetes ([ ]), separando con blancos o comas (,) las columnas y con nueva línea o punto y coma (;) para separar filas. También pueden definirse de forma abreviada usando la notación de dos puntos (:) o las órdenes especiales **linspace** y **logspace**.

Variable matricial = **Inicio:incremento:final**

Si no se pone **incremento** se supone que es **1**

Ej.:

X=0:2:15 equivale a teclear x=[0 2 4 6 8 10 12 14]

X=5:-1.5:1 equivale a teclear x=[5 3.5 2]

X=**linspace(primer valor, último valor, número de valores)**

Ej.: x=linspace(1,10,6) equivale a x=[1.0 2.8 4.6 6.4 8.2 10.0]

Si se desea usar un espaciado logarítmico en lugar de lineal, se puede usar la variante:

X=**logspace(primer exponente, ultimo exponente, número de valores)**

La forma de direccionar los arrays también utiliza la notación de los dos puntos (:), además de la posibilidad de indicar un solo elemento: x(i,j,...) o bien algunos en concreto: x([3 6 1 4]) equivale a un array que contiene los elementos 3, 6, 1 y 4 de x, en ese orden.

I.4.1.-OPERACIONES CON ARRAYS.

Las operaciones en que intervienen arrays pueden ser operaciones de tipo escalar o de tipo matricial. Cuando puede haber confusión, se distingue una de otra poniendo un punto (.) antes del símbolo de la operación correspondiente, para indicar que es una operación escalar. Veamos algunas de las posibilidades:

X+1	Suma 1 a todos los elementos de x
2*X	Multiplica por dos todos los elementos de x
X.*Y	Multiplica cada elemento de X por su homólogo de Y (igual orden)
X./Y	Divide cada elemento de x por su correspondiente de Y (= Y.\X)
X.^2	Eleva cada elemento de X a 2
X.^Y	Eleva cada elemento de X a cada homólogo de Y
X*Y	Multiplicación matricial (ojo con los órdenes)
X/Y	División matricial (= Y\X) (ojo con los órdenes)

Podemos crear arrays columna de dos formas

a)Explícitamente: X=[1; 2; 3; 4; 5]

b)Creando un vector fila y trasponiéndolo sin conjugar (.'):

X=[1 2 3 4 5].'

(nótese que el símbolo ' se reserva para trasponer, mientras que .' se usa para trasponer sin conjugar. Si solamente se actúa sobre reales, ambos hacen lo mismo).

I.5.-GRÁFICAS SIMPLES

Antes de considerar rigurosamente las extensas capacidades gráficas vamos a generar algunas gráficas simples.

Los pasos para representar una gráfica son los dos siguientes:

a)Primeramente hay que generar los arrays correspondientes de abcisas y ordenadas. Por ejemplo:

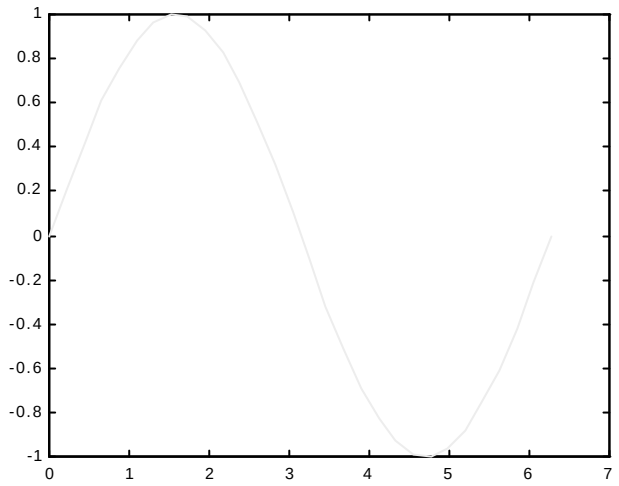
```
x=lispace(0,2*pi,30);
y=sin(x);
```

b)Ahora se usa la orden **plot:**

```
plot(x,y)
```

El resultado puede verse en la siguiente imagen.

MATLAB interpola entre puntos (linealmente), escoge los límites de los ejes. Permite dibujar varias curvas juntas, distintas trazas y colores. Permite representar un vector frente a sus índices, la parte real frente a la compleja (paramétricas), poner una rejilla, ...



I.6.-ARCHIVOS SCRIPT.

Como se habrá comprobado, si se ha de repetir algún cálculo, dar nuevos valores a algunas variables y recalcular todo, etc, puede resultar bastante tedioso, ya que habría que volver a teclear de nuevo todo. Para evitar esto (y por otras razones no menos importantes), MATLAB permite que se puedan colocar órdenes en un fichero de texto (ASCII) y luego decirle que lo abra y lo evalúe exactamente igual que si fuésemos tecleándolo nosotros sobre el espacio de trabajo. Estos ficheros se llaman **archivos script** o **archivos-M** (podemos llamarlos como queramos, pero su extensión debe ser forzosamente **.M**, de ahí su nombre).

Pueden crearse desde cualquier editor de textos (incluido el más cómodo, el Bloc de Notas de Windows, al que se accede desde **F**ile: **N**ew: M-file).

Una vez creado el archivo .M, para ejecutarlo bastará teclear su nombre (ojo: MATLAB buscará primero si, en el actual espacio de trabajo, existe una variable con ese nombre o existe alguna orden propia de MATLAB con ese nombre, si no, entonces buscará un archivo .M en el directorio actual). Una vez llamado, este archivo puede llamar o usar cualquier variable del entorno actual de trabajo y viceversa, es decir, el espacio de trabajo puede acceder a cualquier variable de este fichero (de nuevo, ojo ante posibles variables con el mismo nombre y que, presumiblemente fuesen distintas; esto veremos que no se cumple con las funciones .M).

Si queremos cambiar algún valor del fichero .M, habrá que abrirlo (con **O**pen –M-file) editarlo y volverlo a guardar. Después podremos volverlo a llamar desde el espacio de trabajo.

Estos fichero también son muy útiles para crear grandes arrays (comprobará lo tedioso que es teclear un vector de 10 columnas y, al final, comprobar que se ha equivocado en algún punto anterior: tendrá que volver a teclearlo completamente).

Es muy conveniente introducir comentarios a lo largo de los scripts que documenten adecuadamente la finalidad y funcionamiento del mismo. Además,

recuerde usar adecuadamente los punto y coma (;) para evitar ver resultados intermedios intrascendentes y que embrollan la presentación en pantalla.

MATLAB dispone de algunas órdenes de gestión de archivos .M:

Orden	Operación
what	Devuelve un listado de todos los ficheros .M del directorio actual
dir	Lista todos los archivos del directorio actual
ls	Idéntico a dir
type fichero	Visualiza el fichero .M indicado
delete fichero	Borra el archivo fichero.M (físicamente del dispositivo)
cd camino	Cambia al directorio camino (si no se indica camino dice el actual
chdir camino	Igual que cd camino
pwd	Igual que cd (sin camino)
which fichero	Visualiza el camino de fichero.M

I.7.-CADENAS DE CARACTERES (TEXTO)

A pesar de que la potencia de MATLAB está en los números, también puede tratar con caracteres. Para no variar, los trata como vectores fila.

Hay que encerrarlos entre comillas simples (‘)

```
>>saludo=’Hola, ¿cómo estás?’;
>>x=saludo(13:17)
x=
estás
```

Se puede manipular el vector igual que cualquier array numérico:

```
>>x=saludo(17:-1:13)
x=
sátse
```

Al igual que las matrices, las cadenas pueden tener múltiples filas, pero como aquellas, todas deben tener el mismo número de columnas: habrá que rellenar con blancos para que todas las cadenas tengan la misma longitud).

Las operaciones matemáticas sobre cadenas se realizan sobre el equivalente ASCII de cada carácter:

```
>>letras=’ABCabc’
>>abs(letras)
ans=
65 66 67 97 98 99
>>letras+1
ans=
66 67 68 98 99 100
>>setstr(ans)
```

```
ans=
BCDbcd
```

I.8.-OPERACIONES RELACIONALES Y LÓGICAS

El objetivo de éstos es proporcionar respuestas a cuestiones Verdadero / Falso. Cuando MATLAB encuentra algo verdadero, devuelve un 1 y un 0 en caso contrario

La lista de **operadores relacionales** es:

Operador	Relación
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que
==	Igual a
~=	Distinto a

Pueden utilizarse para comparar dos arrays del mismo tamaño o para comparar un array con un escalar (en este último caso el escalar se compara con cada elemento del array y el resultado tiene el mismo tamaño que el array).

```
>>A=1:5;B=[1,3,5,2,4]
>>A-(B>3)
ans=
1   2  2  4  4
```

(resta 1 a A cuando B es mayor que 3)

```
>>A=[1, 3, 0, 2, 0]
>>A=A+(A==0)*eps
A=
        1.00     3.0000  0.0000  2.0000  0.0000
```

(se han sustituido los ceros por eps =2.2e-16, lo que evitará problemas si A estuviese dividiendo en algún lugar –nos daría un NaN):

```
>>sin(A(3))/A(3)
>>ans=
1
```

La lista **de operadores lógicos** es

Operador	Operación lógica
&	AND
|	OR
~	NOT

Combinando ambos tipos de operadores podemos, por ejemplo, generar arrays que representen señales con discontinuidades o señales que se componen de segmentos. La idea básica es multiplicar aquellos valores que se deseen mantener por 1 y por cero los restantes.

Se dispone además de algunas funciones relacionales y lógicas adicionales:

Función	Descripción
xor(x,y)	Operación or exclusiva (cero cuando, tanto x como y, son cero o uno)
any(x)	Devuelve 1 si algún elemento en un vector es no nulo.
all(x)	Devuelve 1 si todos los elementos de un vector son no nulos.
isnan(x)	Devuelve unos en magnitudes no numéricas en x
isinf(x)	Devuelve unos en magnitudes infinitas en x
finite(x)	Devuelve unos en valores finitos en x

Tabla de preferencias

^	.^	‘	.’		
*	/	\	.*	./	.\
+	-	~	+(unitario)	-(unitario)	
:	>	<	>=	==	~=
|	&				

I.9.-ÁLGEBRA MATRICIAL

Como es sabido, el cálculo matricial proporciona una forma simple y elegante de resolver sistemas de ecuaciones lineales: [b] = [A]· [x]

El caso determinado es cuando el número de incógnitas y de ecuaciones es igual (sin existir entre las primeras combinaciones lineales). En este caso, el determinante de la matriz asociada no debe ser nulo, esto es: **det(A)**, o también **|A|** <>0. En este caso, la solución existe y es única, pudiendo calcularse de una de las dos formas siguientes:

a)x=**inv(**A**)***b

b)x=A\b (esta última es la recomendada, entre otras cosas, por exactitud y rapidez).

Algunas otras operaciones y resultados son los siguientes:

A**.’** Devuelve la traspuesta de A (sin conjugar)
A**’** Da la transpuesta conjugada de A
d=**eig(**A**)** Da los valores propios de A como un vector columna
[v,D]=**eig(**A**)** Devuelve los vectores propios en la matrix V y los valores propios como elementos diagonales en la matriz D.
rank(A**)** Da el rango de A.
poly(A**)** Encuentra el polinomio característico asociado a A.

Estas son solamente algunas de las muchas funciones soportadas por MATLAB para álgebra lineal.

I.10.-MANIPULACIÓN MATRICIAL

MATLAB dispone de una potencia muy elevada para el tratamiento matricial. Vamos a ver solamente algunas de sus más elementales posibilidades.

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

(crea una matriz B tomando las filas de A en orden inverso).

El mismo resultado se obtiene poniendo

```
>>B=A(3:-1:1,:)
```

(Los dos puntos al final significan tomar todas las columnas).

```
>>C=[A  B(:,[1 3])]
c=
        1   2   3   7   9
        4   5   6   4   6
        7   8   9   1   3
```

(crea C añadiendo todas las filas en la primera y tercera columna de B a la derecha de A)

```
>>C=[1 3];
>>B=A(C,C)
B=
        1   3
        7   9
```

(usa el array C para indexar la matriz A)

```
>>B=A(:)
B=
        1
        4
        7
        2
        5
        8
```

```
        3
        6
        9
```
(construye B al disponer A en un vector columna tomando todas sus columnas a un tiempo).

```
>>B=A;
>>B(:,2)=[ ]
B=
        1   3
        4   6
        7   9
```
(redefine B eliminando todas las filas en la segunda columna de la original B. Cuando se fija algo igual a la matriz vacía [ ], lo fijado se suprime, originando que la matriz colapse a lo que permanece>).

```
>>B=A(:[2  2  2  2])
B=
        2   2   2   2
        5   5   5   5
        8   8   8   8
```
(crea B duplicando todas las filas en la segunda columna de A cuatro veces).

También pueden usarse **arrays lógicos 0-1**, si el tamaño del array es igual que el del array que se direcciona. En este caso se retienen los elementos verdaderos (1) y se descartan los elementos falsos (0).

```
>>x=-3:3
x=
        -3   -2   -1   0   1   2   3
>>abs(x)>1
ans=
        1   1   0   0   0   1   1
>>y=x(abs(x)>1)
y=
        -3   -2   2   3
```
(crea y al tomar aquellos valores de x donde su valor absoluto es mayor que 1).

```
>>y=x([1   1   1   1   0   0   0])
y=
        -3   -2   -1   0
>>y=x([1  1  1  1])
y=
        -3   -3   -3   -3
```
(nótese que ahora no se toma el array como lógico, por no tener la misma dimensión que la de x).

MATLAB introduce la función **find**, que devuelve los subíndices donde una expresión relacional es verdadera (también funciona con matrices) :

```
>>k=find(abs(x)>1)
k=
        1   2   6   7
```
(encuentra aquellos subíndices en donde abs(x)>1).

Cuando una función MATLAB devuelve dos o más variables, se encierran mediante corchetes en el lado izquierdo del signo igual:

```
>>A=[1  2  3;  4  5  6;  7  8  9];
>>[i,j]=find(A>5)
i=
        3
        3
        2
        3
j=
        1
        2
        3
        3
```

(los índices almacenados en I y j son los índices asociados respectivamente con la fila y columna donde la expresión relacional es verdadera, esto es, A(i(1),j(1)) es el primer elemento de A donde A>5, ...

En los casos donde se desconoce el tamaño de una matriz, MATLAB proporciona dos funciones de utilidad: **size** y **length**

```
>>A=[1  2  3  4;5  6  7  8];
s=size(A)
s=
        2   4
>>[r,c]=size(A)
r=
        2
c=
        4
>>length(A)
ans=
        4
```

(devuelve el número de filas o el de columnas, el que sea mayor).

Otras características:

flipud(A**)**	intercambia una matriz de arriba abajo.
fliplr(A**)**	intercambia una matriz de izquierda a derecha
rot90(A**)**	gira una matriz en dirección contraria a las agujas del reloj
diag(v**)**	crea una matriz diagonal, con el vector v sobre la diagonal.
diag(A**)**	extrae la diagonal de la matriz A como un vector columna.

Algunas matrices especiales son:

zeros(n**)**	crea una matriz nxn de ceros
ones(n,m**)**	crea una matriz nxm de unos.
rand(nxm**)**	crea una matriz nxm de números aleatorios de distribuidos entre 0 y 1.
eye(n**)**	crea una matriz identidad de orden nxn

I.11.-CONTROLES DE FLUJO

Los lenguajes de programación disponen de órdenes que permiten controlar la secuencia de ejecución, mediante la toma de decisiones. MATLAB también incorpora las más típicas, como son los bucles FOR, WHILE y las estructuras IF-ELSE

I.11.1.-BUCLES FOR

Permiten realizar un conjunto de órdenes un número preestablecido de veces.

for x=array
    órdenes
end

Las órdenes entre las sentencias for y end se ejecutan una vez para cada columna del array.

Un bucle for no puede terminar reasignando a la variable del bucle n un valor dentro del bucle for (cosa que sí se permite en BASIC, por ejemplo).

Cualquier array válido en MATLAB es aceptable en el bucle for.

```
>>for m=1:10
    x(m)=cos(m*pi/10)
end

>>datos=[1 3 5; 8 –2 4]
for n=datos
    x=n(1)-n(2);
end
x=
    -7
x=
    5
x=
    1
```

Los bucles for pueden anidarse como se desee. También deben evitarse siempre que haya un método de array o matriz equivalente que permita resolver un problema dado (por rapidez).

I.11.2.-BUCLES WHILE.

Un bucle WHILE evalúa un grupo de órdenes un número indefinido de veces.

while expresión
    Órdenes
end

Las órdenes entre las sentencias while y end se ejecutan mientras todos los elementos en “expresión” son verdadero.

```
>>num=0;EPS=1;
>>while (1+EPS)>1
        EPS=EPS/2;
        Num=num+1;
end
>>num
num=
        53
EPS=2*EPS
EPS=
        2.2204e-016
```

I.11.3.-ESTRUCTURA IF-ELSE-END.

Cuando una serie de órdenes deben ejecutarse si se cumplen una serie de condiciones, hay que recurrir a este tipo de estructura.

Posibilidades:

a)**if** condición
        órdenes
end

b)**if** condición
        órdenes si condición es verdadero
else
        órdenes si condición es falso
end

c)**if** condición1
        órdenes si condición1 es verdadero
elseif condición2
        órdenes si condición2 es verdadero
elseif ....
....
else
        órdenes si ninguna otra condición es verdadera.
end

La orden final **else** puede no aparecer.

Con esta estructura sí que es posible saltar o romper los bucles for y while: **break**

```
>>EPS=1;
for num=1:1000
EPS=EPS/2
If (1+EPS)<=1
```

```
        EPS=EPS*2
        Break
        end
end

EPS=
        2.2204e-016
```

I.12.-FUNCIONES EN ARCHIVOS –M

MATLAB proporciona una estructura para crear funciones propias en la forma de archivos –M de textos almacenados.

Un archivo –M de función es diferente de un archivo script ya que una función se comunica con el espacio de trabajo de MATLAB sólo a través de las variables pasadas y mediante las variables de salida que genera. Las variables intermedias dentro de la función no aparecen ni interaccionan con el espacio de trabajo de MATLAB.

La primera línea de un archivo –M de función define el archivo –M como una función, especifica su nombre (que debe coincidir con el del fichero en que se guarde) y los nombres de las variables de entrada y salida. Las siguientes líneas de comentarios son el texto visualizado en respuesta a la orden de ayuda **help** (la primera línea de comentarios, conocida como línea H1, es la línea que examina la orden **lookfor).** Finalmente, el resto del archivo –M contiene órdenes MATLAB que crean las variables de salida.

Cuando una función tiene más de una variable de salida, éstas se encierran entre corchetes (Ej.: [v,D]=eig(A)).

La primera vez que MATLAB ejecuta un archivo –M de función abre el correspondiente archivo de texto y compila las órdenes en una representación interna que acelera su ejecución para todas las llamadas subsiguientes.

MATLAB busca los archivos –M de función como los script (primero supone que es una variable, luego una función propia, luego busca en el directorio actual y, por último en el PATH de MATLAB).

El número de variables de entrada pasadas a una función está disponible dentro ,de la función, en la variable **nargin**. El número de variables de salida solicitadas cuando una función se llama está disponible dentro de la función en la variable **nargout**.

Ej.:

```
Function y = fliplr(x)
%FLIPLR Flip matriz in the left / right direction.
% FLIPLR(x) returns X with row preserved and
%  columns flipped in the left / right direction.
%
%  x = 1  2   3   becomes   3   2   1
        4  5  6                  6  5   4
%
%  see also FLUPUD, ROT90
%  Copyright © 1.99¡84-93  by The MathWorks, Inc.
```

```
[m,n]=size(x);
y=x(:,n:-1:1);
```

I.13.-ANÁLISIS DE DATOS

Por convenio, los conjuntos de datos se almacenan en matrices orientadas por columnas, esto es, cada columna representa una variable y cada fila representa muestras individuales de las variables.

Ej.: Si temps son tres columnas de 31 datos cada una (temperatura de tres ciudades todos los días de Julio), y definimos d como una fila: d=1:31, la orden plot(d,temps) representará esas tres curvas (columnas) de variación de temperatura con los 31 días de abcisa (fila).

mean(temps**)** daría un vector fila de tres valores (las medias de cada columna).

Si la entrada a una función de análisis de datos es un vector fila o columna, MATLAB simplemente realiza la operación sobre el vector, devolviendo un resultado escalar.

Análogamente ocurre con **max** y **min**, que si se llaman con dos variables, también indicarán el "día en que se producen" (realmente el índice)

Función	Descripción
corrcoef(x)	Coeficiente de correlación
cov(x)	Matriz de covarianza
cumprod(x)	Producto acumulativo de columnas
cumsum(x)	Suma acumulativa de columnas
dic(x)	Calcula las diferencias entre elementos
hist(x)	Histograma o diagrama de barras
mean(x)	Valor medio de columnas
median(x)	Valor de la mediana de columnas
prod(x)	Producto de elementos en columnas
rand(x)	Números aleatorios distribuidos uniformemente
randn(x)	Números aleatorios distribuidos normalmente
sort(x)	Ordenar columnas en orden ascendente
std(x)	Desviación estándar de columnas
sum(x)	Suma de elementos en cada columna

I.14.-POLINOMIOS

Se dan igual que un vector fila: P = [1 -12 0 15 116] equivale al polinomio:

$x^4-12x^3+25x+116$

Se pueden obtener las raíces de un polinomio mediante la orden r = **roots(**P**)** y viceversa, obtener un polinomio a partir de sus raices: PP=**poly(**r**)** .Suele ser interesante usar, tras esta orden, la orden **real(**PP**)** para pasar a real (eliminar partes imaginarias).

La multiplicación polinomial se realiza mediante la convolución: c=**conv(**a,b**)** (para más productos hay que hacerlo sucesivamente).

La suma se realiza como simple suma de vectores (la pega es que hay que ponerlos de la misma longitud).

La división se realiza mediante la orden: [q,r]=**deconv(**c,b**)** (q es el cociente y r el resto).

Se pueden derivar polinomios: h=**polyder(**g**)** . Para derivar un cociente de polinomios también se puede usar esa misma orden: **polyder(**n,d**)** deriva n/d

Para evaluar un polinomio se usa **polyval(**p,x**)**:

```
>>x=linspace(-1,2);P=[1  4  -7  -10];
>>v=polyval(P,x)
```

Por último, una interesante función es **residue(**n,d**)**, que calcula el desarrollo en fracciones simples el cociente n/d (n y d son polinomios).

I.15.-AJUSTE DE CURVAS E INTERPOLACIÓN

I.15.1.-AJUSTE DE CURVAS.

Se puede ajustar por mínimos cuadrados (dando los datos y el orden del polinomio que deseamos que mejor ajuste) usando la orden **polyfit(**x,y, n**)**. Para n=1, se ajusta a una recta, n=2 una parábola, etc. Da como resultado un vector fila con los coeficientes del polinomio obtenido.

I.15.2.-INTERPOLACIÓN UNIDIMENSIONAL.

Devuelve el valor de la curva interpoladora (recta si no se indica nada, o polinomios de tercer orden si se indica la palabra clave 'spline'):

-t=**interp1(**x,y,valor_deseado**)**

Realiza una interpolación lineal. Valor_deseado (que también puede ser un array) es el valor en el que queremos saber el resultado de la interpolación.

-t=**interp1(**x,y,x_0,**'spline')**

I.15.3.-INTERPOLACIÓN BIDIMENSIONAL.

Interpola funciones de dos variables. Existen tres posibles métodos (si no se indica ninguno se supone que es lineal): **linear, cubic, nearest**:

-z=**interp2(**x, y, z, xi, yi, **'metodo')**.

En este caso, x e y son las dos variables independientes y z es una matriz de la variable dependiente, que tiene el tamaño length(y) filas y length(x) columnas, xi es un array de los valores para interpolar a lo largo del eje x; yi es un array de valores para interpolar a lo largo del eje y.

I.16.-ANÁLISIS NUMÉRICO

I.16.1.-REPRESENTACIÓN GRÁFICA.

A veces es interesante representar una función (no sus valores, como hace la función plot). Para ello MATLAB facilita otra orden llamada **fplot**:

fplot(función,rango**)**.

Se permiten varias posibilidades para indicar la función:

a)Dar, entre comillas simples el nombre de una función de fichero –M: fplot('humps',[0 2]).

b)Indicar expresamente la función: fplot('2*exp(-x).*sin(x)',[0 8])

c)Definir previamente f: f='2*exp(-x).*sin(x)' ; fplot(f,[0 8])

I.16.2.-MINIMIZACIÓN DE FUNCIONES.

Existen dos funciones (**fmin** y **fmins**) que encuentran mínimos de funciones unidimensionales y n-dimensionales, respectivamente (para máximos se usará fmin de la función –f(x)). Funciona de una forma parecida a fplot en la manera de indicar la función. Como es obvio, habrá que indicar entre qué intervalo se debe busvar el mínimo:

Xmin=**fmin(**'funcion',intervalo**)**

Ej.: fn='2*exp(-x).*sin(x)'; xmin=fmin(fn,2,5)

Se puede evaluar una función en un punto usando x=xmin; ymin= **eval(**fn**)**

I.16.3.-LOCALIZACIÓN DE CEROS.

Para esta orden, solamente se puede usar el apartado a) para describir la función (archivo –M): **fzero('**humps**',**1.2**)** busca un cero de la función 'humps' en los

alrededores de 1.2. Si queremos resolver f(x)=c, hay que usar esta función sobre g(x)=f(x)-c.

I.16.4.-INTEGRACIÓN NUMÉRICA.

Hay tres métodos posibles: **trapz** (método trapezoidal), **quad** y **quad8** (enfoque de cuadraturas, obtienen resultados más precisos y se llaman de manera similar a fzero).

trapz(x,y**)**

quad('humps',-1,2**)**

I.16.5.-DERIVACIÓN NUMÉRICA.

Aunque no debe usarse por los problemas asociados (sería preferible interpolar y derivar), MATLAB permite la derivación numérica: dy = **diff(**y**)./diff(**x**)**

I.16.6.-ECUACIONES DIFERENCIALES (RESOLUCIÓN NUMÉRICA).

Se dispone de dos métodos: **ode23** (recomendable cuando se desean más puntos de salida) y **ode45** (más rápida para pocos puntos).

Primeramente habrá que poner la ecuación original como un vector de ecuaciones de primer orden La solución vendrá dada por un vector columna de derivadas, solución del archivo –M correspondiente.

Ej.:

```
Function yprime=vdpol(t,y);
%VDPOL(t,y) returns the state derivative of the
%Van Der Pol equation:
%
%   x''-mu*(1-x^2)*x'+x=0  ('=d/dt   ''=d^2(dx^2)
%
% let y(1)=x   and   y(2)=x'
%
%  then  y(1)'=y(2)
%          y(2)'=mu*(1-y(1)^2)*y(2)-y(1)
%
mu=2;   %cose  0 < mu < 10
yprime=[y(2)
          mu*(1-y(1)^2)*y(2)-y(1)];   %yprime is a column
```

(nótese que la variable escalar t no se usa, pero es necesario definirla.

Dada esta función la solución se calcula como:

```
[t,y]=ode23('vdpol',0,30,[1;0]);
y1=y(:1);
y2=y(:2);
plot(t,y1,t,y2,'—')
```

La forma general es **ode23(**funcion,t0,tf,y0**)**

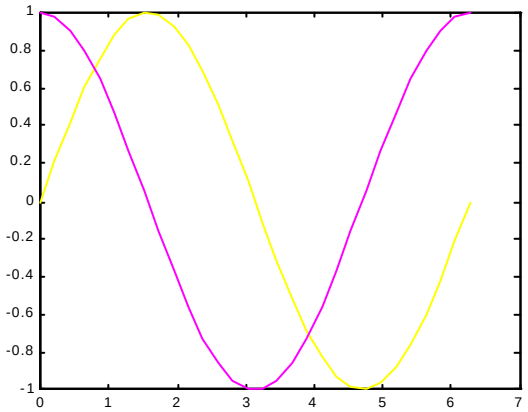
Siendo t0 el tiempo inicial, tf el final e y0 un vector columna de condiciones iniciales).

I.17.-GRÁFICOS 2-D

I.17.1.-UTILIZACIÓN DE LA ORDEN PLOT.

Ya hemos usado **plot(**v_indep,v_dep**)**. Pero se dispone de más utilidades, por ejemplo, se pueden poner varias parejas de variables, y representar así, varias curvas simultáneamente, es más, si uno de los argumentos es una matriz y el otro un vector, la orden plot representa cada columna de la matriz respecto del vector:

```
>>x=lispace(0,2*pi,30);y=sin(x);z=cos(x);
>>plot(x,y,x,z)
```

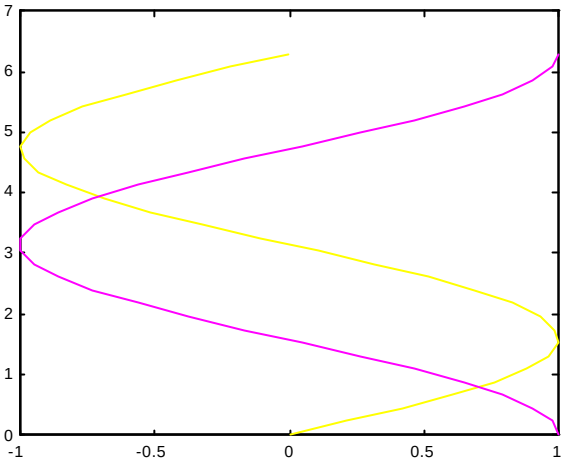


También produciría el mismo resultado la secuencia:

```
>>W=[y ; z];plot(x,W)
```

Si se cambia el orden de los argumentos, la gráfica girará 90º:

>>plot(W,x)



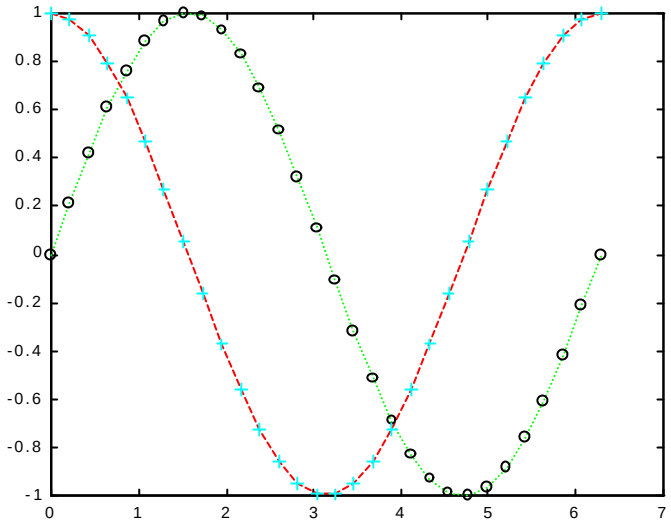
I.17.2.-ESTILO DE LÍNEAS, MARCADORES Y COLORES.

Se pueden especificar los colores y los estilos de línea que queramos, dando un argumento adicional a plot, después de cada pareja de arrays de datos. El argumento opcional es una cadena de 1, 2 ó 3 caracteres de la tabla siguiente:

Símbolo	Color	Símbolo	Estilo de línea
y	amarillo	.	punto
m	magenta	o	círculo
c	cian	x	marca –x
r	rojo	+	más
g	verde	*	estrella
b	azul	-	línea sólida
w	blanco	:	línea punteada
k	negro	-.	Línea punto-raya
		--	Línea a trazos

Si no se especifica un color, MATLAB comienza con el amarillo y va cambiando a través de los primeros seis colores de la tabla para cada línea adicional. El estilo de línea por defecto es la línea sólida a menos que especifique un estilo de línea diferente. La utilización de los símbolos punto, círculo, marca-x y estrella coloca el símbolo escogido en cada punto, pero no los conecta con una línea recta (obviamente, estos colores no se mostrarán en esta impresión).

>>plot(x,y,'g:',x,z,'r—',x,y,'wo',x,z,'c+')

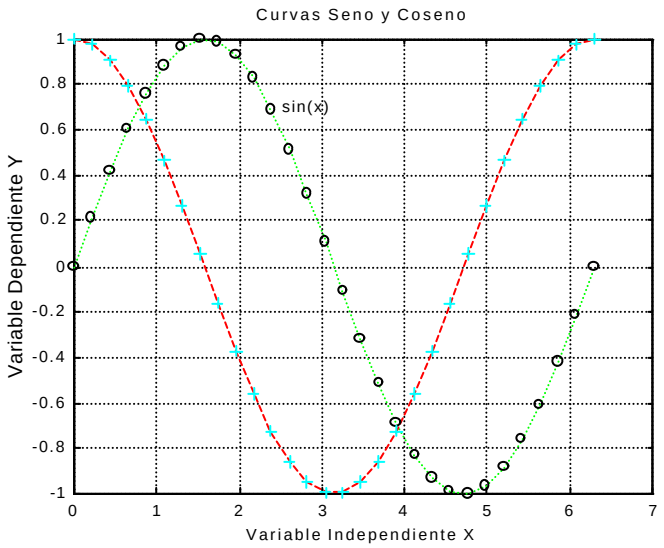


I.17.3.-ADICIÓN DE REJILLAS Y ETIQUETAS.

La orden **grid on** añade una rejilla a la gráfica actual. La orden **grid off** elimina la rejilla, **grid** sin ningún argumento alternativamente se activa o desactiva.

Los ejes horizontal y vertical se pueden etiquetar respectivamente con las órdenes **xlabel** e **ylabel**. La orden **title** añade una línea de texto en la parte superior de la gráfica.

```
>>grid
>>xlabel('Variable Independiente X')
>>ylabel('Variable Dependiente Y')
>>title('Curvas Seno y Coseno')
```



También se puede añadir una etiqueta o cualquier otra cadena de texto a cualquier localización específica en su gráfica, con la orden **text(**x, y, 'string'**)**, donde (x,y) representa las coordenadas de la arista del entro izquierda de la cadena de texto en unidades tomadas de los ejes de la gráfica.

```
>>text(2.5,0.7,'sin(x)')
```

Si no queremos pararnos a analizar las coordenadas, se puede colocar una cadena de texto con el ratón. La orden **gtext** conmuta la ventana de la figura actual, pone una marca en cruz que sigue el ratón y espera una pulsación del ratón o de una tecla. Cuando una u otra acción ocurre, el texto se coloca con la esquina inferior izquierda del primer carácter en esa localización: gtext('sin(x)').

I.17.4.-EJES A MEDIDA.

Si no está satisfecho con el escalado y apariencia de ambos ejes, podemos cambiarlos mediante la orden **axis**. Esta orden es muy extensa, y para estudiarla con detenimiento se recomienda usar la ayuda (help axis).

I.17.5.-IMPRESIÓN DE FIGURAS.

Para imprimir una gráfica podemos ir a la ventana correspondiente y seleccionar PRINT del menú File, pero MATLAB también tiene sus propias órdenes para estos efectos. Para imprimir una ventana de figura, pulse sobre ella con el ratón o use la orden **figure(**n**)** para hacerla activa y luego utilice la orden **print**.

La orden **orient** cambia el modo de orientación: el modo por defecto es el modo retrato que imprime verticalmente en el centro de la página, el modo paisaje imprime horizontalmente y llena la página. El modo tall imprime verticalmente pero llena la página. El modo de impresión elegido permanece inalterado hasta que se cambie o finalice su sesión de MATLAB.

I.17.6.-MANIPULACIÓN DE GRÁFICOS.

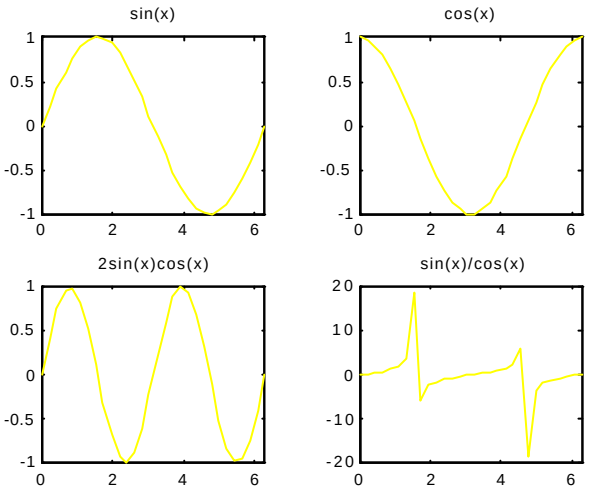
Puede añadir líneas a una gráfica existente usando la orden **hold**. Cuando fija **hold on** MATLAB no elimina las curvas existentes cuando se emiten las nuevas órdenes plot. En su lugar, añade nuevas curvas a los ejes actuales. Sin embargo, si los nuevos datos no se ajustan dentro de los límites de los ejes actuales, los ejes se reescalan. Fijando **hold off** se libera la ventana de la figura actual para una nueva gráfica. La orden **hold** sin argumento conmuta el valor de hold.

Si necesita dos o más gráficas en diferentes ventanas de figuras, use la orden **figure** en la ventana de orden o de figura; sin ningún argumento crea una nueva ventana de figura. Se puede elegir una ventana de figura seleccionándola con el ratón o usando figure(n).

Una ventana de figura puede mantener más de un conjunto de ejes. La orden **subplot(m,n,p)** subdivide la ventana de la figura actual en una matriz mxn de las áreas de representación gráfica y escoge como activa el área p-ésima. Ls subgráficas se numeran de izquierda a derecha a lo largo de la fila superior, luego la segunda fila, etc.

```
>>x=linspace(2, 2*pi,30);
>>y=sin(x);
```

```
>>z=cos(x);
>>a=2*sin(x).*cos(x);
>>b=sin(x)./(cos(x)+eps);
>>subplot(2,2,1)
>>plot(x,y),axis([0 2*pi –1 1]), title('sin(x)')
>>subplot(2,2,2)
>>plot(x,z),axis([0 2*pi –1 1]), title('cos(x)')
>>subplot(2,2,3)
>>plot(x,a), axis([0 2*pi –1 1]), title('2sin(x)cos(x)')
>>subplot(2,2,4)
>>plot(x,b),axis([0 2*pi –20 20]),title('sin(x)/cos(x)')
```



Use **subplot(1,1,1)** para retornar al modo por defecto y utilizar la ventana de figura para un único conjunto de ejes

MATLAB proporciona una herramienta interactiva para expandir secciones de un gráfico 2-D para verlas con más detalle o para ampliar una región de interés. La orden **zoom on** activa el modo zoom. Cada vez que se pulse el botón derecho se expande la gráfica por un factor 2 centrado alrededor del punto debajo del ratón (con el botón derecho se comprime un factor 2). También se puede pulsar y arrastrar para ampliar un área específica. **zom out** devuelve la gráfica a su estado inicial. **Zoom** sin argumentos conmuta el estado del zoom de la ventana de la zona activa.

I.17.7.-OTRAS CARACTERÍSTICAS DE LOS GRÁFICOS 2-D.

loglog es lo mismo que plot excepto que se usan escalas logarítmicas para ambos ejes.

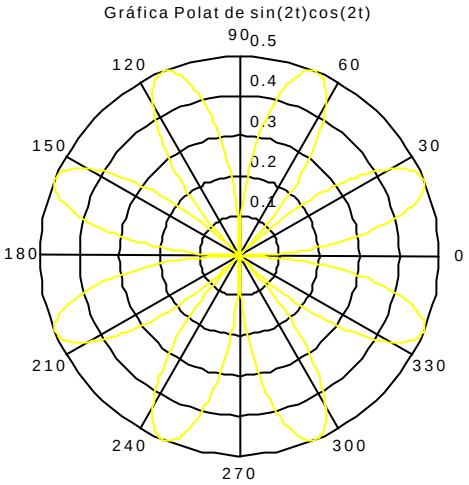
semilogx es lo mismo que plot, excepto que usa escala logarítmica en el eje x y escala lineal en el eje y.

semilogy es lo mismo que plot excepto que usa escala lineal en el eje x y escala logarítmica en el eje y.

Se pueden crear gráficos en coordenadas polares utilizando la orden **polar(t,r,S)**, donde t es el vector de ángulos en radianes, r es el radio vector y S es

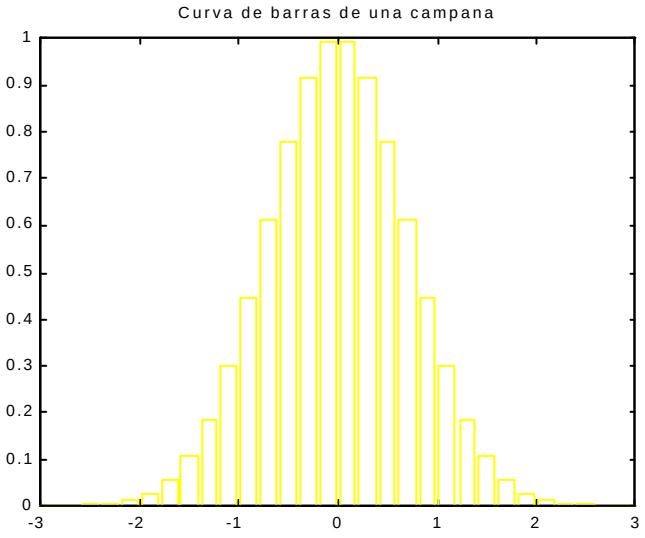
una cadena de caracteres opcional que describe color, símbolo que se emplea para marcar y/o estilo de línea.

```
>>t=0:0.01:2*pi;
>>r=sin(2*t).*cos(2*t);
>>polar(t,r)
>>title('Gráfica Polar de sin(2t)cos(2t)')
```

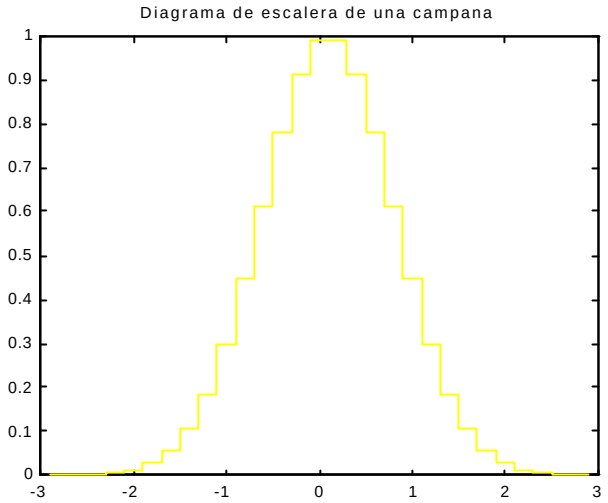


Los gráficos de barras y escaleras se pueden generar usando las órdenes gráficas **bar** y **stairs**. Veamos unos ejemplos:

```
>>x=-2.9:0.2:2.9;
>>y=exp(-x.*x);
>>bar(x,y)
title('Curva de barras de una campana')
```




```
>>stairs(x,y)
>>title('Diagrama de escalera de una campana')
```

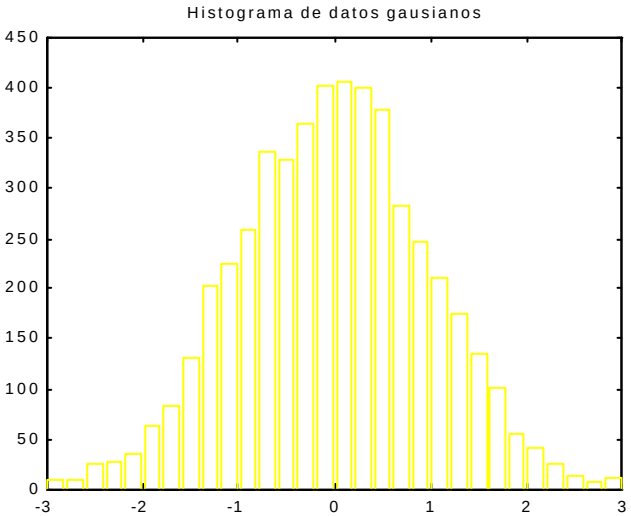


También podemos obtener histogramas con las tres siguientes posibilidades:

a)**hist(**y**)** dibuja un histograma con diez elementos para los datos en el vector y.
b)**hist(**y,n**)** , donde n es un escalar, dibuja un histograma con n elementos.
c)**hist(**y,x**)**, donde x es un vector, dibuja un histograma utilizando los elementos especificados en x.

Veamos un ejemplo de un histograma de una curva en forma de campana de datos gaussianos:

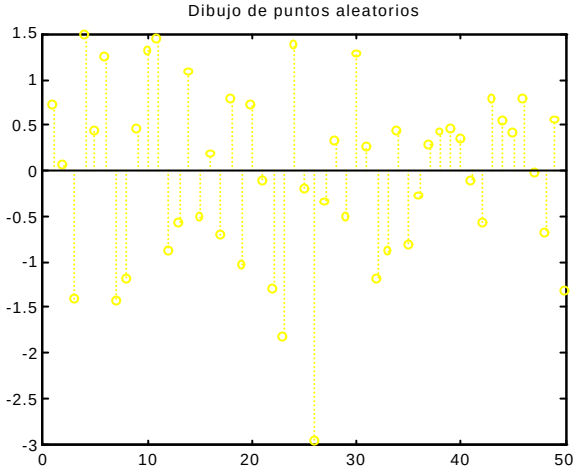
```
>>x=-2.9:0.2:2.9;
>>y=randn(5000,1);
>>hist(y,x)
>>title('Histograma de datos gaussianos')
```



Una secuencia de datos discretos se puede representar usando la función **stem(y)**, que crea una gráfica de los valores en el vector y conectado al eje horizontal por una línea. Un argumento opcional, que es una cadena de caracteres, se puede utilizar para especificar el estilo de línea. **Stem(**x,y**)** representa los puntos en y en los valores especificados en x.

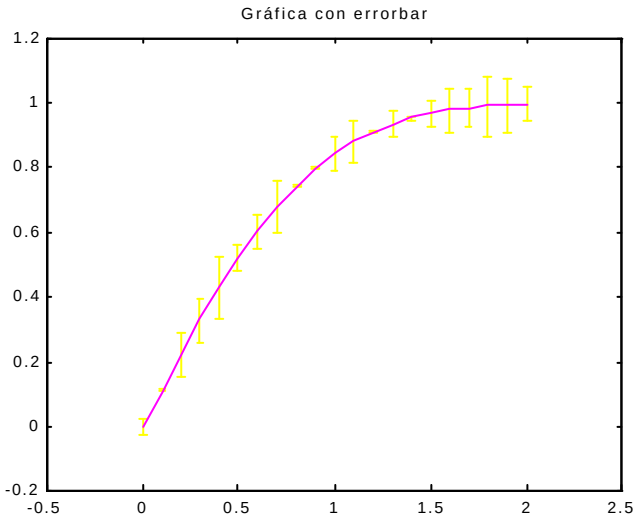
Ej.:

```
>>y=randn(50,1);
stem(y,':');
title('Dibujo de puntos aleatorios')
```

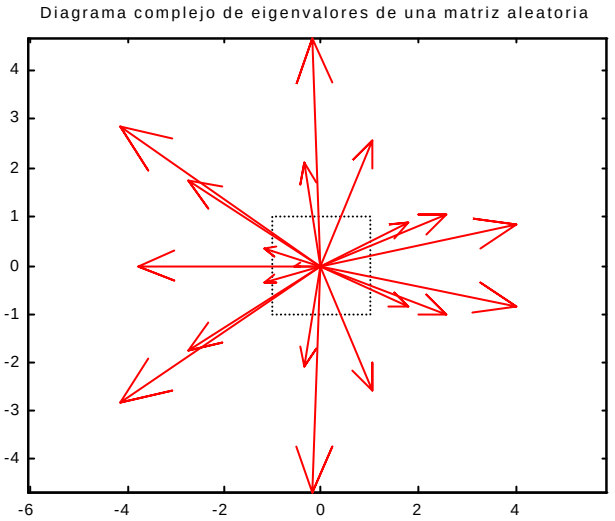


Una gráfica puede incluir barra de error en los puntos.: **errorbar(**x,y,e**)** representa la gráfica del vector x frente al vector y con barras de error especificado por el vector e. Todos los vectores deben tener la misma longitud. Para cada punto (x,y) se dibuja una barra de error una distancia e, por encima, y e, por debajo del punto.

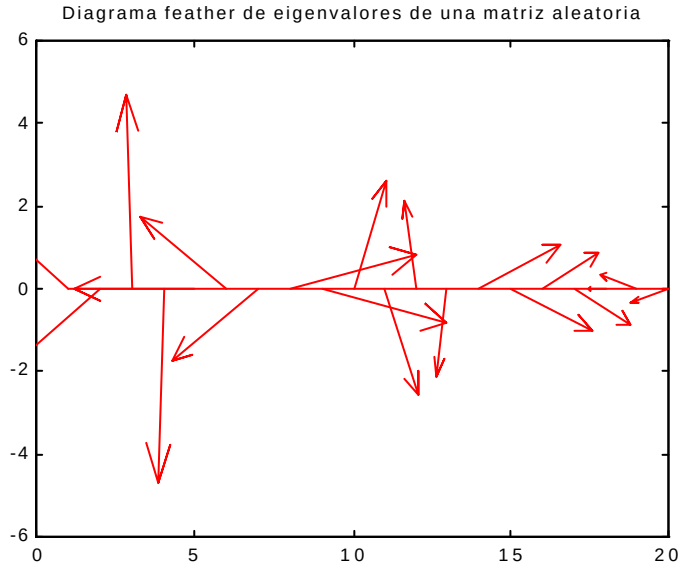
```
>>x=0:0.1:2;
>>y=erf(x);
>>e=rand(size(x))/10;
>>errorbar(x,y,e)
>>title('Gráfica con errorbar')
```



Se pueden representar datos complejos usando **compass** y **feather**:
a)**compass(z)** dibuja una gráfica que visualiza el ángulo y la magnitud de los elementos complejos de z como flechas que emanan del origen.
b)**feather(z)** representa los mismos datos usando flechas que emanan de puntos igualmente espaciados sobre una línea horizontal.
compass(x,y**)** y **feather(**x,y**)** son equivalentes a **compass(**x+i*y**)** y **feather(**x+i*y**)**.

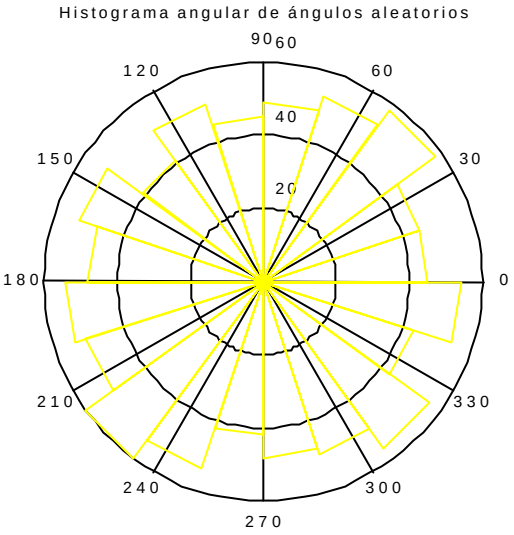


La figura anterior y la siguiente se obtienen de la siguiente forma:
>>z=eig(randn(20,20);
>>compass(z)
>>title('Diagrama complejo de eigenvalores de una matriz aleatoria')
>>feather(z)
>>title('Diagrama feather de eigenvalores de una matriz aleatoria')



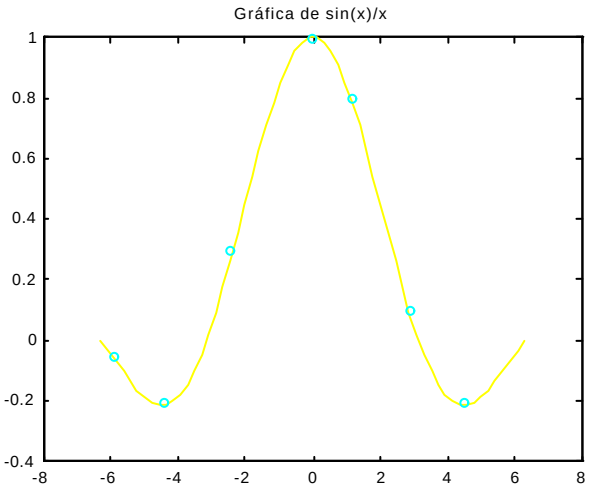
rose(t**)** dibuja un histograma polar de 20 elementos para los ángulos del vector t. **rose(**t,n**)**, donde n es un escalar, dibuja un histograma con n elementos. **rose(**t,x**)**, donde x es un vector, dibuja un histograma usando los elementos especificados en x. Ej.:

>>t=randn(1000,1)*pi;
>>rose(t)
>>title('Histograma angular de ángulos aleatorios')



ginput proporciona un medio de seleccionar puntos de una gráfica usando el ratón: **[**x,y**]=ginput(**n**)** toma hasta n puntos del eje actual y devuelve sus coordenadas en los vectores columna x e y. Si se omite n se adquieren un número ilimitado de puntos hasta que se pulsa la tecla ENTER.

>>x=linspace(-2*pi,2*pi,60);
>>y=sin(x)./(x+eps);
>>plot(x,y)
>>title('Gráfica de sin(x)/x')
>>[a,b]=ginput(8);
>>hold onn
>>plot(a,b,'co')
>>hold off

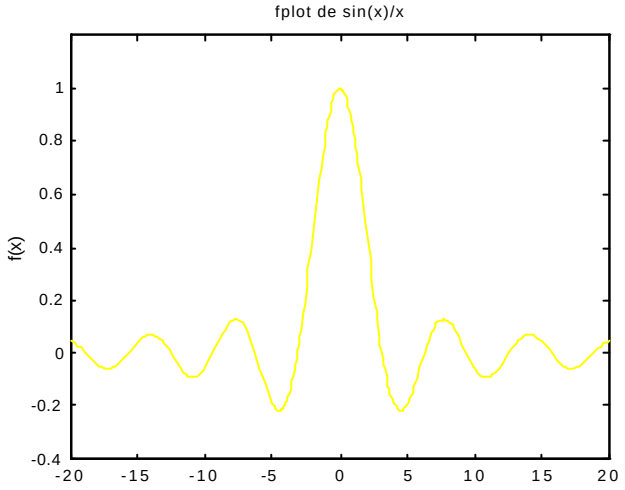


La orden **fplot** da una forma fácil de representar automáticamente una función de una variable entre límites especificados, sin crear previamente un conjunto de datos para la variable: **fplot(‘**función**’,[**xmin xmax**])** representa la función en el rango xmin, xmax con escalado automático del eje y. **Fplot(‘**función**’,[**xmin xmax ymin ymax**])** especifica también los límites del eje. Existen restricciones sobre el tipo de función que se puede representar y se pueden especificar argumentos adicionales.

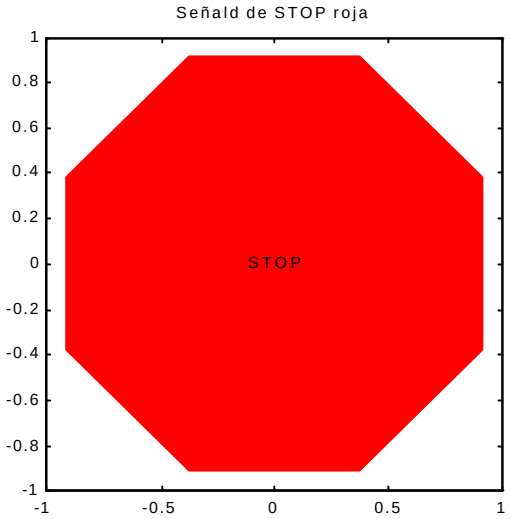
```
>>fplot(‘sin(x)./x’),[-20 20 -.4 1.2])
>>title(‘fplot de sin(x)/x’)
>>ylabel(‘f(x)’)
```

El resultado puede apreciarse en la figura siguiente.

Por último, la orden **fill(**x,y,’c’**)** rellena el polígono 2-D definido por los vectores columna x e y con el color especificado por c. Los vértices del polígono se especifican por los pares (xi, yi). Si es necesario, el polígono se cierra conectando el último vértice con el primero.




```
>>t=(1/8:2/8:15/8)’*pi  %vector columna
>>x=sin(t);
>>y=cos(t);
>>fill(x,y,’r’)
>>axis(‘square’)
>>text(-.11,0,’STOP’)
>>title(‘Señal de STOP roja’)
```



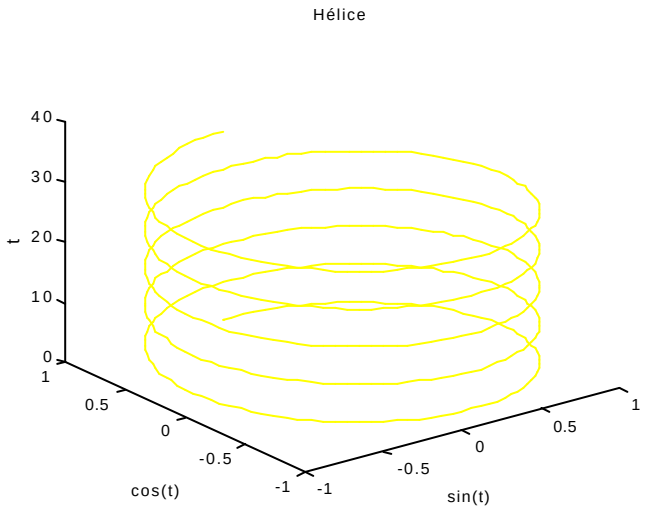
I.18.-GRÁFICOS 3-D

MATLAB proporciona una variedad de funciones para visualizar datos 3-D. Algunas funciones representan líneas en tres dimensiones, mientras que otras dibujan superficies. Se puede emplear el pseudocolor para representar una cuarta dimensión.

I.18.1.-GRÁFICOS DE LÍNEA.

La orden **plot3** es la generalización a 3-D de la orden plot de 2-d. El formato es el mismo, excepto que los datos están en tripletes, en lugar de aparecer por parejas.
Plot3(x1,y1,z1,s1,x2,y2,z2,s2,...**)**, donde x_n, y_n, y z_n son vectores o matrices y s_n son cadenas de caracteres opcionales que especifican color de marcado y/o estilo de línea. Veremos como ejemplo una hélice:

```
>>t=0:pi/50:10*pi;
>>plot3(sin(t),cos(t),t)
>>title('Hélice'),xlabel('sin(x)),ylabel('cos(x)'),zlabel('t')
```



Al igual que existe la función **zlabel** para 3-D, también se tiene una versión de axis: **axis([**xmin xmax ymin ymax zmin zmax**])** fija los límites de los tres ejes. Análogamente **text(**x, y, z, 'string'**)** coloca el texto 'string' comenzando por la coordenada (x,y,z) sobre la gráfica actual.

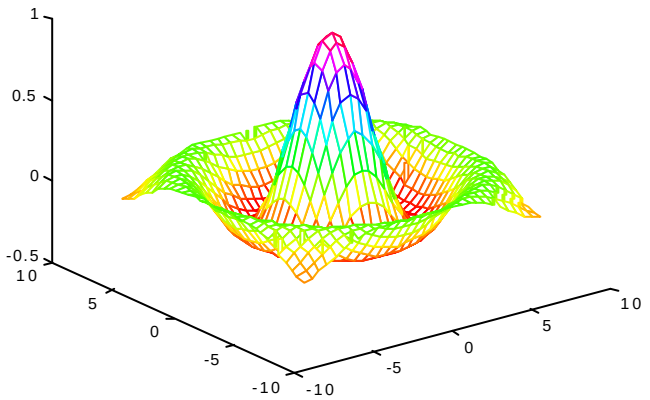
I.18.2.-GRÁFICOS DE MALLA Y DE SUPERFICIE.

MATLAB define una superficie de malla por las coordenadas z de los puntos que están por encima de una rejilla rectangular en el plano xy. Se forma una gráfica uniendo puntos adyacentes con líneas rectas. El resultado parace como una red de pescar con los nudos en los puntos. Los gráficos de mallas son muy útiles para visualizar grandes matrices o representar funciones de dos ariedades.

El primer paso para generar una gráfica de malla de una función de dos variables z=f(x,y), es engendrar matrices x e y formadas, respectivamente, por filas y columnas repetidas, en algún rango de las variables x e y. MATLAB proporciona la función **meshgrid** para este objetivo. **[X,Y]=meshgrid(**x,y**)** crea una matriz X cuyas filas son copias del vector x, y una matriz Y cuyas columnas son copias del vector y. Este par de matrices se pueden utilizar para evaluar funciones de dos variables utilizando las características matemáticas de los arrays en MATLAB.

Ej.:

```
>>x=-7.5:.5:7.5;
>>y=x;
>>[X,Y]=meshgrid(x,y);
>>R=sqrt(X.^2+Y.^2)+eps;
>>z=sin(R)./R;
>>mesh(X,Y,Z)
```



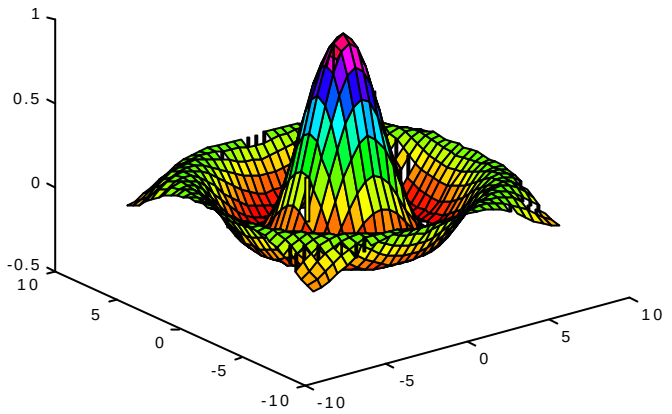
Obsérvese que los colores de línea están relacionados con la altura de la malla por encima del plano XY. **mesh** aceptará un argumento opcional para controlar los colores y los rangos de color usados en la gráfica.

Si se usa con un solo argumento: **mesh(Z)** utilizará los puntos (i, j, Z_{ij}), esto es, se representa Z frente a sus subíndices.

Una gráfica de superficie de la misma matriz Z parece como la gráfica de malla previamente generada, excepto que se rellenan los espacios entre las líneas

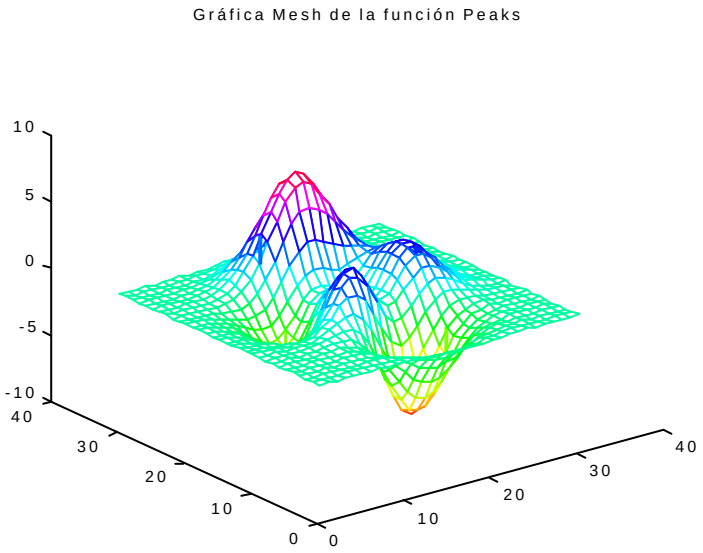
(llamados parches). Las gráficas de este tipo son generadas usando la función **surf**, que tienen los mismos argumentos que la función mesh:

>>surf(X,Y,Z)



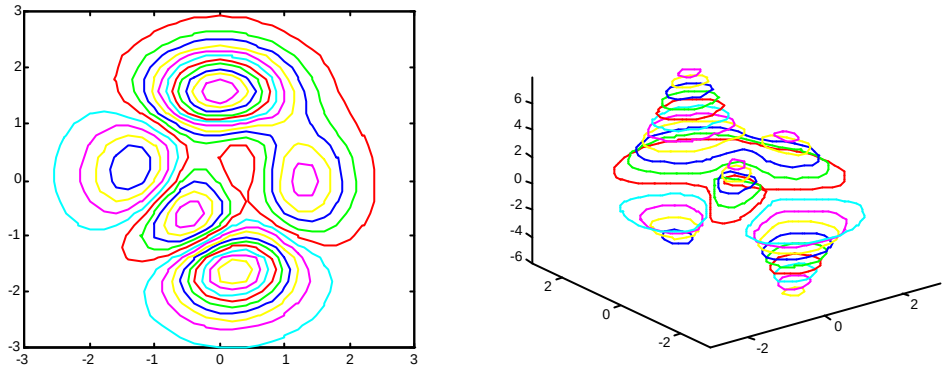
Para los siguientes ejemplos vamos a usar la función **peaks**:

>>mesh(peaks)
>>title('Gráfica Mesh de la función Peaks')



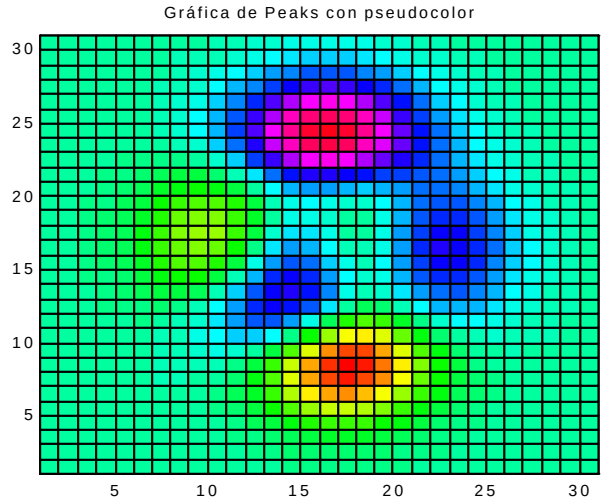
Las gráficas de contorno muestran líneas de elevación o altura constante (especie de mapas topográficos). En MATLAB, las gráficas de contorno en 2-D y 3-D se generan usando, respectivamente, las funciones **contour** y **contour3**.

>>[x,y,z]=peaks;
>>contour(x,y,z,20) %genera 20 contornos en 2-D
>>contour3(x,y,z,20) %idem, en 3-D
>>axis([-3 3 -3 3 -6 8])



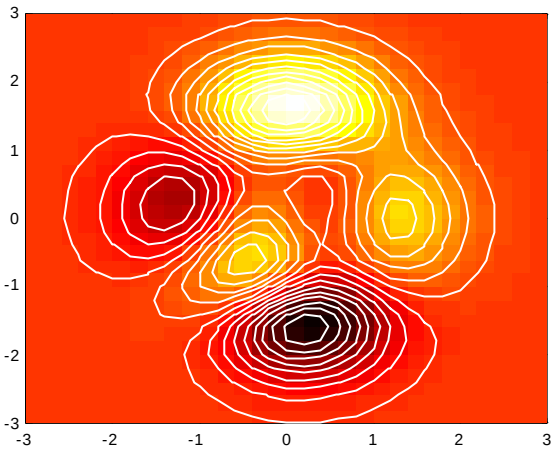
Otra manera de visualizar información de contorno es empleando color para representar la altura. La función **pcolor** transforma la altura a un conjunto de colores y presenta la misma información que la gráfica contour en la misma escala.

```
>>z=peaks;
>>pcolor(z)
>>title('Gráfica de Peaks con pseudocolor')
```



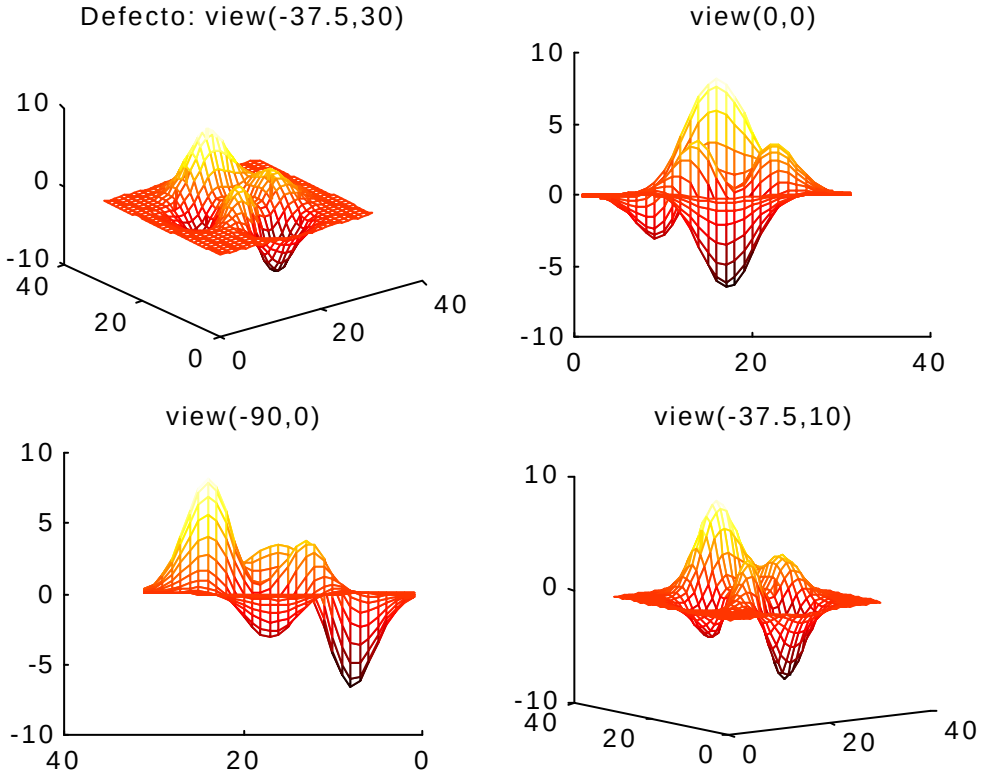
Como pcolor y contour dan la misma información, con la misma escala, a veces es conveniente representarlas conjuntamente. Veremos un ejemplo donde se usan, además, **colormap** y **shading** para cambiare la apariencia de la gráfica.

```
>>[x,y,z]=peaks;
>>colormap(hot)  %selecciona un conjunto de colores
>>pcolor(x,y,z)
>>shading flat  %elimina las líneas de la rejilla
>>hold on
>>contour(x,y,z,20,'k')
>>hold off
```



I.18.3.-MANIPULACIÓN DE GRÁFICOS.

MATLAB permite especificar el ángulo desde el que se ve una gráfica 3-D. La función **view(**azimut, elevación**)** fija el ángulo de visión especificando el azimut (ángulo dentro del plano xy donde se encuentra, se mide en grados desde el eje negativo y) y la elevación (ángulo en grados desde el que se observa la gráfica por encima del plano xy, 30º por defecto).



La orden view tiene otra forma que puede ser más útil en algunas circunstancias: **view([**x y z**])** coloca su vista en un vector que contiene las coordenadas cartesianas (x,y,z) en el espacio 3-D. La distancia a la que se está del

origen no afecta. Además, el azimut y la elevación de la vista actual se puede obtener utilizando **[**az,el**]=view**

La orden **hidden** controla la eliminación de líneas escondidas. Puede elegirse entre tres tipos de sombreado para las gráficas mesh, surf, pcolor y fill: sombreado plano (**shading flat**), interpolado (**shading interp**) y por caras (**shading faceted**), que es el que se da por defecto.

Existen otras muchas características que, por su extensión, no vamos a ver aquí (relativas al color y los llamados mapas de color). Se recomienda acudir a libros especializados para su análisis detallado.

BLOQUE II
TUTORIAL DEL TOOLBOX DE MATEMÁTICA SIMBÓLICA

II.1.-INTRODUCCIÓN

MATLAB básico carece de la posibilidad de manipular expresiones matemáticas sin usar realmente números, mediante el toolbox de matemática simbólica se puede hacer que MATLAB manipule expresiones que se calculan como símbolos matemáticos, en lugar de operar con números. En este conjunto de herramientas (tools) existe una gran variedad, que permiten combinar, simplificar, diferenciar, integrar y resolver ecuaciones algebraicas y diferenciales. Otras herramientas son usadas en álgebra lineal para obtener resultados exactos de inversas, determinantes y formas canónicas; así como encontrar autovalores de matrices simbólicas sin el error introducido por los cálculos numéricos. También se soporta aritmética de precisión variable que se calcula simbólicamente y devuelve un resultado con una precisión especificada.

Las herramientas del toolbox de matemática simbólica están construidas sobre el potente programa de software llamado MAPLE.

II.2.-EXPRESIONES SIMBÓLICAS

Las expresiones simbólicas son cadenas de caracteres, o arrays de cadenas de caracteres, que representan números, funciones, operadores y variables. Las variables no necesitan tener valores predefinidos. Las ecuaciones simbólicas son expresiones simbólicas conteniendo un signo igual. La aritmética simbólica es el ejercicio de resolver estas ecuaciones simbólicas aplicando reglas e identidades conocidas a los símbolos dados. Las matrices simbólicas son arrays cuyos elementos son expresiones simbólicas.

II.2.1.-REPRESENTACIÓN EN MATLAB DE EXPRESIONES SIMBÓLICAS

MATLAB representa internamente las expresiones simbólicas como cadenas de caracteres, para diferenciarlas de las variables numéricas o de las operaciones.

A veces basta con poner la expresión entre comillas simples(‘’), pero en otros casos hay que usar explícitamente el comando **sym**: M=sym(‘[a,b;c,d]’). Si se puede, es equivalente poner:

diff cos(x) = diff(‘cos(x)’) = diff(sym(‘cos(x)’))

(ojo: si no se ponen las comillas hay que tener cuidado con los espacios en blanco).

Las expresiones simbólicas sin variables son llamadas **constantes simbólicas**. Por ejemplo, f = symop(‘(3*4-2)/5+1’) da como resultado el carácter 3 (si sumamos 1 a f sale 52, que es el código ASCII de 3 más 1).

II.2.2.-VARIABLES SIMBÓLICAS.

Cuando se trabaja con expresiones simbólicas conteniendo más de una variable, sólo una es la variable independiente. Si no se le dice a MATLAB cuál es la variable independiente, se selecciona basándose en la siguiente regla:

"*Es la única letra minúscula, distinta de i y j, que no es parte de una palabra. Si no existe tal carácter, se elige x. Si el carácter no es único, se elige el más cercano alfabéticamente a x. Si se produce un empate, se elige el carácter posterior según el alfabeto*".

La orden **symvar(‘**expresión**’)** devuelve la variable independiente escogida por MATLAB. La variación **symvar(‘**expresión**’,’**carácter**’)** encuentra la variable más próxima al carácter dado (si no se pone se supone que carácter es x).

Ej.:

symvar(‘a*x+y’) de cómo resultado x
symvar(‘a*t+s/(u+3)’) da como resultado u
symvar(‘sin(omega)’) da como resultado x
symvar(‘3*i+4*j’) da como resultado x
symvar(‘y+3*s’,’t’) encuentra la variable más próxima a t, o sea, s.

Muchos comandos dan la opción de especificar la variable independiente, por ejemplo: diff(‘x^n’<,’n’>).

II.3.-OPERACIONES SOBRE EXPRESIONES SIMBÓLICAS

Si la expresión es un polinomio racional (cociente de dos polinomios), es frecuente que necesitemos separar el numerador y el denominador. Esto se puede hacer con la orden **[**n,d**]=numden(‘**expresión**’)**. Devuelve el numerador en n y el denominador en d. Si expresión es un array se devuelven dos arrays, uno (n) con los numeradores y otro (d) con los denominadores. Si solamente se pone n=numdem(‘expresión’), sólo se devuelve el numerador.

Si la expresión contiene sumas, éstas se realizan previamente.

II.3.1.-OPERACIONES ALGEBRAICAS ESTÁNDAR.

Las más utilizadas son:

Suma:	**symadd(**f,g**)**	f+g
Resta:	**symsub(**f,g**)**	f-g
Multiplicación:	**symmul(**f,g**)**	f*g
División:	**symdiv(**f,g**)**	f/g
Potenciación:	**sympow(**f,’3*x’**)**	f^{3x}
General:	**symop(**f,’/’,g,’+’,3)	f/g+3

Esta última acepta hasta 16 argumentos separados por comas, cada uno de los cuales puede ser una expresión simbólica, un valor numérico o un operador. Symop concatena los argumentos y devuelve la expresión resultante.

II.3.2.-OPERACIONES AVANZADAS.

Veremos en este apartado cómo MATLAB permite la composición de funciones y la obtención de la inversa de una función, entre otras.
Si tenemos las siguientes funciones:

$f = \frac{1}{1+x^2}$; $g = \sin(x)$; $h = \frac{1}{1+u^2}$; $k = \sin(v)$

Composición de funciones:

compose(f,g**)** obtiene $f(g(x)) = \frac{1}{1+[\sin(x)]^2}$

compose(g,f) obtiene $\sin\left(\frac{1}{1+x^2}\right)$

Si tiene otras variables independientes hay que indicarlo:

Compose(h,k,'u','v') da $h(k(v)) = \frac{1}{1+\sin(v)^2}$

Función inversa: **finverse(‘**expresión**’<,’**variable**’>)**.
finverse(‘x^2’) da $\sqrt{x}$
finverse(‘a*b+c*d-a*z’,’a’) da -(c*d-a)/(b-z)

Suma simbólica: **symsum(**función<,’variable’,min,max>**)**

symsum(f) da $\sum_{0}^{x-1} f(x)$

symsum(f,’s’) da $\sum_{0}^{s-1} f(s)$

symsum(f,a,b) da $\sum_{a}^{b} f(x)$

symsum(f,’s’,a,b) da $\sum_{a}^{b} f(s)$

Funciones de conversión: Son contrarias a sym:

a)**numeric(**‘expresión’**)**: convierte una expresión simbólica sin variables (lo que hemos llamado una constante simbólica) a su valor numérico.
phi=‘1+sqrt(5))/2’; numeric(phi) da 1.6180

b)**eval(**‘expresión**’)** Pasa una cadena de caracteres a MATLAB para evaluarla: eval(phi) también devuelve 1.6180

c)**sym2poly**(‘expresión’**)** convierte un polinomio simbólico en su vector equivalente de MATLAB. **poly2sym(**vector<,’variable’>**)** hace lo contrario:

```
>>f='2*x^2+x^3-3*x+5'
f=
        2*x^2+x^3-3*x+5
>>n=sym2poly(f)
n=
        1   2  -3  5
```

```
>>poly2sym(n)
ans=
        2*x^2+x^3-3*x+5
>>poly2sym(n,'s')
ans=
        s^3+2*s^2-3*s+5
```

d)**Sustitución de variables**: Si queremos cambiar una variable por otra, podemos usar la orden **subs(**f,'var1','var2').

Si f='a*x^2+b*x+c' y g='3*x^2+5*x-4':

subs(f,'s','x') cambia x por s: a*s^2+b*s+c
h=subs(g,'2','x') da como resultado el string h=18

e)**Racionalización** de expresiones simbólicas: **symrat(**matriz escalar**)** obtiene una expresión simbólica racional de esa matriz escalar.

II.4.-DIFERENCIACIÓN E INTEGRACIÓN

II.4.1.-DIFERENCIACIÓN.

Existen las siguientes posibilidades:

a)**diff(**f**)** obtiene la derivada de f con respecto a la variable por defecto.
b)**diff(**f,'var'**)** deriva f respecto de la variable 'var'.
c)**diff(**f,2**)** derivada segunda de f respecto a la variable por defecto.
d)**diff(**f,'var',2**)** derivada segunda de f respecto de la variable var.

La función diff también opera sobre arrays, derivando cada elemento de la matriz simbólica.

(recuérdese que diff también es usado para derivación numérica, si el argumento no es simbólico).

II.4.2.-INTEGRACIÓN.

Si MATLAB no encuentra solución, devuelve el comando sin evaluar.

Tiene seis posibilidades de ser invocada:

a)**int(**f**)** obtiene $\int f dx$

b)**int(**f,'s'**)** obtiene $\int f ds$

c)**int(**f,a,b**)** obtiene $\int_a^b f dx$

d)**int(**f,'s',a,b**)** obtiene $\int_a^b f ds$

e)**int(**f,'m','n'**)** obtiene $\int_m^n f dx$ (siendo m y n variables simbólicas).

f)**int(**f,'s','m','n'**)** obtiene $\int_m^n f ds$ (siendo m y n variables simbólicas).

(Igual que con diff, int realiza la integración numérica si los argumentos no son simbólicos).

II.5.-REPRESENTACIÓN GRÁFICA DE EXPRESIONES SIMBÓLICAS

Se puede representar una función simbólica en el rango –2pi, 2pi con escalado automático mediante la orden: **ezplot(**'expresion'**)**, a la vez que añade automáticamente cuadrícula y etiqueta la gráfica. También podemos escoger nosotros el rango:
Ezplot('expresión',[a b]**)**.

Por supuesto, también le son de aplicación los comandos de dibujo ya vistos: axis, title, xlabel, ylabel, ...

II.6.-FORMATEADO Y SIMPLIFICACIÓN DE EXPRESIONES

Cuando el resultado de MATLAB sea difícil de leer, podemos probar con diversos comandos para hacer el resultado más legible.

Supondremos para los ejemplos que: f=sym('(x^2-1)*(x-2)*(x-3)')

a)**Visualización tradicional**: **pretty('**expresión**')**. Este comando intenta visualizar una expresión simbólica en una forma que se parece a las matemáticas que figuran en los libros de texto. Pruébese, por ejemplo:

```
>>g=int('log(x)/exp(x^2)')
g=
        int(log(x)/exp(x^2),x)
>>pretty(g)
                    /
                    |  log(x)
                    |---------- dx
                    |         2
                    | exp(x  )
                    /
```

b)**Agrupamiento de términos**: La orden **collect(f)** daría como resultado:
x^4-5*x^3+5*x^2+5*x-6

c)**Representación anidada de Horner**: **horner(f)** da como resultado:
-6+(5+(5+(-5+x)*x)*x)*x

d)**Factorización**: **factor(**ans**)** nos volvería a dar f

e)**Expansión de productos en sumas**: La orden **expand(f)** expande los productos en sumas. El resultado sería: x^4-5*x^3+5*x^2+5*x-6

f)**Simplificación**: Hay dos opciones.

f1)**simplify(**'expresión'**)** es una herramienta muy potente que involucra muchas clases de identidades. Por ejemplo, simplify('log(2*x/y)') da como resultado log(2) + log(x) –log(y)

f2)**simple(**'expresión'**).** Es una de las más potentes, pero la menos ortodoxa. Prueba varias formas de simplificar y elige la que menos caracteres necesita para representar el resultado (también permite ver el resultado de cada prueba que hace).

Ej.:

Si $f = \sqrt[3]{\frac{1}{x^3}+\frac{6}{x^2}+\frac{12}{x}+8}$

```
>>simple(f)
simplify:
        (2*x+1)/x
ans=
        (2*x+1)/x
>>simple(ans)
combine(trig):
        2+1/x
ans=
        2+1/x
```

Otra herramienta que puede ser de interés, si se utiliza el programa LATEX es **latex**, que presenta varias opciones.

Las funciones simbólicas de MATLAB se pueden combinar para convertir una expresión simbólica a su representación en fracciones simples: dada una función f, cociente de dos polinomios racionales, int(f) integrará la función y, por regla general, separará los términos. Entonces, diff(ans) diferenciará cada término para producir la expresión original f en forma de suma de términos que es la representación en fracciones parciales de f.

II.7.-ARITMÉTICA DE PRECISIÓN VARIABLE

En los valores numéricos de cualquier precisión se puede introducir un error de redondeo, sin embargo, las operaciones sobre expresiones simbólicas son altamente precisas ya que no se realizan cálculos numéricos y no hay errores de redondeo. La precisión relativa de las operaciones aritméticas individuales de MATLAB es alrededor de 16 dígitos. Por otro lado, las capacidades simbólicas de MAPLE pueden realizar operaciones con un número arbitrario de dígitos (lo que conlleva más tiempo y espacio de almacenado). La precisión de MAPLE es de 16 dígitos, pero se puede variar usando **digits(**n**)**, donde n es el número deseado de dígitos de precisión. La función **digits** sin argumento devuelve el valor actual del parámetro global Digits. Hay que indicar que, aunque se cambie la precisión, la visualización de los resultados no cambia.

Se dispone de otra función que permite realizar un cálculo simple con una precisión arbitraria, sin modificar el parámetro global Digits: **vpa(**'expresión',n**)** siendo n el número de dígitos significativos con los que se evalúa la expresión simbólica y también, los utilizados para la visualización.

Ej: vpa('pi',20) da como resultado 3.1415926535897932385

También puede aplicarse a una matriz simbólica.

II.8.-RESOLUCIÓN DE ECUACIONES

II.8.1.-RESOLUCIÓN DE UNA ECUACIÓN ALGEBRAICA SIMPLE.

La orden adecuada es **solve(**'expresión'<,'var'>**)** . Si la expresión no contiene ningún signo igual, MATLAB supone que es igual a cero. Si hay varias soluciones, éstas se ponen en un vector.

Los resultados pueden evaluarse con la orden ya vista numeric. Si no puede encontrar una solución simbólica, entonces devuelve una de precisión variable.

Ej.: solve('a*x^2+b*x+c') devuelve un vector simbólico con las soluciones

[1/2/a*(-b+(b^2-4*a*c)^(1/2)]
[1/2/a*(-b-(b^2-4*a*c)^(1/2)]

A veces, si el número de soluciones es muy elevado, para abreviar su visualización MATLAB no las imprime todas, e indica la expresión **'Root of (expresión)'** . En este caso y si realmente queremos verlas todas, habrá que indicárselo mediante la orden **allvalues(**sol**)**.

II.8.2.-RESOLUCIÓN DE VARIAS ECUACIONES ALGEBRAICAS SIMPLES.

Podemos resolver varias ecuaciones algebraicas simples al mismo tiempo de la forma:
a)**solve(**e1, e2, ..., en**)** resuelve n ecuaciones para la variable por defecto.
b)**solve(**e1, e2, ..., en,'v1, v2, ..., vn'**)** resuelve n ecuaciones simbólicas para las n incógnitas listadas como 'v1, v2, ..., vn'.

II.8.3.-ECUACIÓN DIFERENCIAL SIMPLE.

La función **dsolve** calcula las soluciones simbólicas de ecuaciones diferenciales ordinarias. Estas ecuaciones se especifican usando el operador diferencias D para indicar la diferenciación (D2, D3, ...). Cualquier letra seguida de Dn es una variable dependiente.

Ej.: $d^2y/dx^2=0$ se podrá como D2y=0

La variable independiente se puede especificar o elegir por defecto según la regla de symvar.

Ej.: Resolver $dy/dx=1+y^2$: **dsolve(**'Dy=1+y^2') devuelve -tan(-x+C1)
Las constantes de integración las denomina C1, C2, ... de forma automática.

También se puede incluir la condición inicial, por ejemplo y(0)=1:

Ej.: Y=dsolve('Dy=1+y^2','y(0)=1') cuyo resultado es tan(x+1/4*pi)
También podemos indicar la variable independiente:

dsolve('Dy=1+y^2','y(0)=1','v') siendo el resultado: tan(v+1/4*pi)

Veamos algunos ejemplos más:

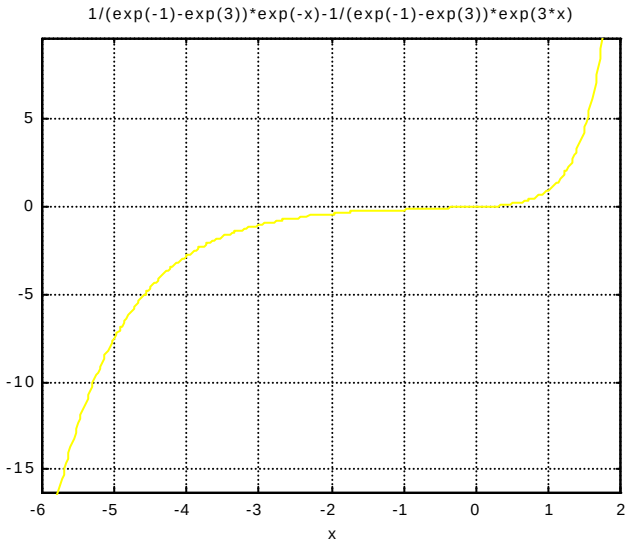
a)Para la ecuación diferencial $\frac{d^2y}{dx^2} = \cos(2x) - y$ con las condiciones iniciales: y(0)=1; y'(0)=0 se resolverá de la siguiente forma:

```
>>y=dsolve('D2y=cos(2*x)-y','Dy(0)=0,y(0)=1')
y=
        -2/3*cos(x)^2+1/3+4/3*cos(x)
>>y=simple(y)
y=
        -1/3*cos(2*x)+4/3*cos(x)
```

b)Para la ecuación $\frac{d^2y}{dx^2} - 2\frac{dy}{dx} - 3y = 0$ con las condiciones y(0)=0, y(1)=1:

```
>>y=dsolve('D2y-2*Dy-3*y=0','y(0)=0,y(1)=1')
y=
        1/(exp(-1)-exp(3))*exp(-x)-1/(exp(-1)-exp(3))*exp(3*x)
>>y=simple(y)
y=
        -(exp(-x)-exp(3*x))/(exp(3)-exp(-1))
>>pretty(y)
                        exp(-x) – exp(3 x)
                -       ------------------------
                        exp(3) – exp(-1)
```

```
>>ezplot(y,[-6 2])
```



II.8.4.-SISTEMAS DE ECUACIONES DIFERENCIALES.

Igual que para ecuaciones algebraicas, se puede resolver un sistema de ecuaciones diferenciales de la forma: **dsolve(**ec1, ec2, ..., ecn<,'k1, k2, ..., kn'>).

Por ejemplo, para resolver $\frac{df}{dx} = 3f + 4g$; $\frac{\mathrm{dg}}{\mathrm{dx}} = -4f + 3g$ con las condiciones iniciales f(0)=0 y g(0)=1

```
>>[f,g]=dsolve('Df=3*f+4*g','Dg=-4*f+3*g','f(0)=0,g(0)=1')
f=
        exp(3*x)*sin(4*x)
g=
        exp(3*x)*cos(4*x)
```

Si x fuese una variable dependiente, se usaría t como independiente.

II.9.-ÁLGEBRA LINEAL Y MATRICES

II.9.1.-MATRICES SIMBÓLICAS.

Son arrays cuyos elementos son expresiones simbólicas. Se pueden generar con la función **sym**, así como dando valores individuales. Veamos algunos ejemplos:

```
>>A=sym('[a,b,c;b,c,a;c,a,b]')
A=
        [a, b, c]
        [b, c, a]
        [c, a, b]
>>G=sym('[cos(t),sin(t);-sin(t),cos(t)]')
G=
        [cos(t), sin(t)]
        [-sin(t), cos(t)]
```

De forma individual (ahora i y j son índices, no la unidad imaginaria)

```
>>S=sym(3,3,'(i+j)/(i-j+s)')
S=
        [     2/s, 3/(-1+s), 4/(-2+s)]
        [3/(1+s),       4/s, 5/(-1+s)]
        [4/(2+s), 5/(1+s),       6/s]
```

Si queremos usar otros índices, habría que indicarlos:

```
>>S=sym(3,3,'m','n','(m-n)/(m-n-t)')
```

La función sym también puede convertir una matriz numérica a su forma simbólica si los elementos se pueden poner como cociente de enteros pequeños se hace, si no, se ponen como números en coma flotante de forma simbólica.

El tamaño (número de filas y columnas) de una matriz simbólica se pueden obtener usando **symsize**, que devuelve un valor numérico, no una expresión simbólica. Puede usarse en cualquiera de sus cuatro variedades:
(usaremos, como ejemplo S=sym('[a,b,c;d,e,f]')

a)d=**symsize(S)** devuelve el vector d = [2 3]
b)[m,n]=**symsize(**S**)** devuelve M=2 y n=3
c)m=**symsize(**S,1**)** da el número de filas: m=2
d)n=**symsize(**S,2**)** devuelve el número de columnas.

Los arays numéricos usan la forma N(m,n) para acceder a un elemento simple, pero los elementos de los arrays simbólicos deben ser referenciados usando funciones simbólicas, tal como **sym(**S,m,n**)**.

Ej.: Sea G=sym('[ab,cd; ef,gh]')

Si llamamos a G(1,2) obtendríamos como respuesta el carácter 'a' (segundo carácter de la primera fila, ya que la matriz simbólica se almacena en el computador como un array de caracteres 2x7: *[ab,cd* es la primera fila y *ef,gh]* la segunda).

Para hacer referencia a la segunda expresión de la primera columna debemos usar: r=**sym(**G,1,2**)**, que nos dará r=cd (expresión simbólica).

También se puede usar para cambiar valores:

```
>>sym(G,2,2,'pq')
ans=
        [ab, cd]
        [ef, pq]
```

II.9.2.-OPERACIONES CON MATRICES SIMBÓLICAS.

Son las siguientes:

a)**Suma 1**:	**symadd(**G,'t'**)**	Añade 't' a cada elemento.
b)**Suma 2**:	**symadd(**G,D**)**	G+D
c)**Resta**:	**symsub(**G,D**)**	G-D
d)**Multiplicación**:	**symmul(**G,G**)**	G*G
e)**Potenciación**:	**sympow(**G,2**)**	G^2
f)**División**:	**symdiv(**G,D**)**	G/D
g)**Transpuesta**:	**transpose(**G**)**	G'
h)**Determinante**:	**determ(**G**)**	|G|
i)**Inversa**:	**inverse(**G**)**	G^{-1}

La resolución de sistemas de ecuaciones lineales (equivalente al numérico \) también puede hacerse con la siguiente orden:

linsolve(A,B**)** resuelve la ec. Matricial A*X = B para una matriz cuadrada A

II.9.3.-OTRAS CARACTERÍSTICAS.

Se pueden concatenar argumentos y evaluar la expresión resultante mediante la orden **symop**. Veamos un ejemplo:

```
>>f='cos(x)'
f=
        cos(x)
>>symop('atan(',f,'+',a,')','^2')
ans=
        atan(cos(x)+a)^2
```

La función **charpoly** obtiene el polinomio característico de una matriz:

```
>>G=sym('[1,1/2;1/3,1/4]')
G=
        [  1,1/2]
        [1/3,1/4]
>>charpoly(G)
ans=
        x^2-5/4*x+1/12
```

Los autovalores y autovectores de las matrices simbólicas se pueden encontrar usando la función **eigensys**:

```
>>F=sym('[1/2,1/4;1/4,1/2]')
F=
        [1/2,1/4]
        [1/4,1/2]
>>eigensys(F)
ans=
        [3/4]
        [1/4]
>>[V,E]=eigensys(F)
V=
        [-1, 1]
        [ 1, 1]
E=
        [1/4]
        [3/4]
```

Pueden obtenerse igualmente la matriz de Jordan, la base del Núcleo de un espacio vectorial, los valores singulares de una matriz, y el jacobiano.

II.10.-TRANSFORMADAS

II.10.1.-FUNCIONES IMPULSO Y SALTO.

MATLAB dispone de dos funciones muy importantes en ingeniería: la función escalón, salto o de Heaviside y la función impulso o delta de Dirac:

a)Función escalón:

$$f(t) = \begin{cases} 0 & ; \ t < a \\ K & ; \ t >= a \end{cases}$$

```
>>u='k*Heaviside(t-a)
```

b)Función impulso. Es la derivada de la función salto

```
>>d=diff(u)
d=
```

k***Dirac(**t-a**)**

II.10.2.-LA TRANSFORMADA DE LAPLACE.

La transformación de Laplace, de gran interés en ingeniería:

$$F(s)=\int_0^{\infty} f(t)e^{-st}dt \qquad ; \qquad f(t)=\frac{1}{2\pi i}\int_{C+j0}^{C+j\infty} F(s)e^{st}ds$$

también es suministrada por MATLAB mediante las órdenes:

F=**laplace(f)** y f=**invlaplace(**F**)**

La variable por defecto de f es t y de F es s. La transformada de Laplace usa la forma symvar(f,'t'). Si no se especifica la variable independiente, la transformada inversa usa symvar(F,'s')

II.10.3.-LA TRANSFORMADA DE FOURIER.

La transformada de Fourier se utiliza ampliamente en Electricidad para determinar características de circuitos.

$$F(w)=\int_{-\infty}^{+\infty} f(t)e^{-jwt}dt \quad ; \quad f(t)=\frac{1}{2\pi}\int_{-\infty}^{+\infty} F(w)e^{jwt}dw$$

Las órdenes que se utilizan son:

F=**fourier(**f<,'w','t'>**)** y f=**invfourier(**F**)**

II.10.4.-LA TRANSFORMADA Z.

Las transformadas de Laplace y de Fourier son usadas para analizar sistemas continuos. Las transformadas z, por otro lado, son usadas para analizar sistemas discretos.

$F(z)=\sum_{n=0}^{\infty} f(n)z^{-n}$ donde z es un número complejo.

En MATLAB, las órdenes pertinentes son:

G=**ztrans(**f**)** y g=**invztrans(**G**)**

Las variables por defecto en las transformadas z son n y z.

Hay que advertir que en todos los pares de transformadas, tienen formas que permiten especificar diferentes variables independientes (se ha indicado en algún caso).

Finalmente, es interesante una herramienta interactiva que simula una calculadora muy completa: **funtool**.

BLOQUE III
TUTORIAL DEL TOOLBOX DE SEÑALES Y SISTEMAS

III.1.-INTRODUCCIÓN

Los sistemas continuos (analógicos) y discretos (digitales) se pueden representar en MATLAB como funciones de transferencia, ganancia-cero-polo, o formas en el espacio de estados. Como las funciones de transferencia se describen como cociente de polinomios, en MATLAB son descritas como vectores fila de sus coeficientes. En la forma ganancia-cero-polo, las raíces de los polinomios del numerador, del denominador y la ganancia global son descritas como arrays columnas. La forma de espacio de estados se utiliza mediante las matrices A, B, C y D.

En el caso continuo (para el discreto solamente habría que sustituir la variable s por la variable z) se tendrá:

Función de transferencia:

$$H(s)=\frac{N(s)}{D(s)}=\frac{N_1 s^m+N_2 s^{m-1}+...+N_m s+N_{m+1}}{D_1 s^n+D_2 s^{n-1}+...+D_n s+D_{n+1}} \quad \text{(con m<=n)}$$

en MATLAB sería: num=[N_1 N_2 ...N_{m+1}], den=[D_1 D_2 ...D_{n+1}]

Ganancia-cero-polo:

$$H(s)=\frac{N(s)}{D(s)}=K\frac{(s-z_1)(s-z_2)...(s-z_m)}{(s-p_1)(s-p_2)...(s-p_n)} \quad \text{(con m<=n)}$$

en MATLAB sería: K, Z=[z_1;z_2;...;z_m] P=[p_1; p_2;...;p_n]

Espacio de estados:

$$\left.\begin{aligned}\dot{x}&=Ax+Bu\\ y&=Cx+Du\end{aligned}\right\}$$

en MATLAB: A, B, C, D.

Esta última opción se utiliza en el caso de múltiples entradas y múltiples salidas (MIMO), mientrs que las dos primeras solamente se utilizan en sistemas de simple entrada y múltiple salida (SIMO). Cuando un sistema tiene múltiples salidas, el denominador de su función de transferencia permanece el mismo para todas las salidas, pero tiene un numerador diferente asociado con cada salida. En MATLAB el numerador es representado mediante una matriz con la i-ésima fila conteniendo el numerador asociado con la i-ésima salida.

III.2.-ORGANIZACIÓN DEL TOOLBOX DE SEÑALES Y SISTEMAS

Se dispone de una gran cantidad de funciones. Para ayudar a encontrar las funciones asociadas con un tipo dado de problema, éstas se organizan en las siguientes tablas. La primera columna de cada tabla es el nombre de la función genérica de MATLAB. La segunda columna muestra las opciones típicas de sintaxis.

RESPUESTA EN FRECUENCIA		
bode	bode(num,den) bode(num,den,W) bode(A,B,C,D,iu) bode(A,B,C,D,iu,W)	Diagrama de Bode
freqs	[H,W]=freqs(num,den) H=freqs(num,den,W) [H,W]=freqs(num,den,n)	Respuesta en frecuencia continua
freqz	[H,W]=freqz(num,den) [H,W]=freqz(num,den,n) H=freqz(num,den,W) [H,f]=freqz(num,den,n,Fs)	Respuesta en frecuencia discreta
nyquist	nyquist(num,den) nyquist(num,den,W) nyquist(A,B,C,D,iu) nyquist(A,B,C,D,iu)	Diagrama de Nyquist

ANÁLISIS EN EL DOMINIO DE LA FRECUENCIA		
abs	abs(x)	Magnitud de una variable compleja
angle	angle(x)	Ángulo de fase de una variable compleja
grpdelay	[Gd,W]=grpdelay(num,den) [Gd,W]=grpdelay(num,den,n) Gd=freqz(num,den,W) [Gd,f]=freqz(num,den,n,Fs)	Retardo de grupo discreto
psd	psd(x) psd(x,nfft) psd(x,nfft,Fs) psd(x,nfft,Fs,window)	Estimación de la densidad espectral de potencia
specgram	specgram(x)	Espectrograma
unwrap	unwrap(x)	Respuesta de fase desplegada

DISEÑO DE FILTROS PROTOTIPO
Butterworth, Chebishev y elípticos,. Consultar Manual

DISEÑO DE FILTROS
Butterworth, Chebyshev, elípticos, FIR, IIR, … Consultar Manual

TRANSFORMACIÓN EN EL DOMINIO DE LAS FRECUENCIAS
PasaBaja/PasaAlta PasaBaja/EliminaBanda, ... Consultar Manual

TRANSFORMADA RÁPIDA DE FOURIER		
fft	fft(x) fft(x,n)	Transformada rápida de Fourier
fft2	fft2(X) fft2(X,nr,nc)	Transformada rápida de Fourier 2-D
ifft	ifft(x) ifft(x,n)	Transformada rápida inversa de Fourier
ifft2	ifft2(X) ifft2(X,nr,nc)	Transformada rápida inversa de Fourier 2-D

FUNCIONES DE VENTANA
Para filtros digitales. Consultar Manual.

CONVERSIÓN CONTINUO A DISCRETO
Consultar Manual.

REPRESENTACIÓN DE SISTEMAS LINEALES		
residue	[R,P,k]=residue(num,den) [num,den]=residue(R,P,k)	Desarrollo en fracciones parciales continua e inversa
ss2tf	[num,den]=ss2tf(A,B,C,D,iu)	Espacio de Estados a Función de Transfer.
ss2zp	[Z,P,K]=ss2zp(A,B,C,D,iu)	Espacio de Estados a zero-polo
tf2ss	[A,B,C,D]=tf2ss(num,den)	Función de Transfer. a Espacio de Estados
tf2zp	[Z,P,k]=tf2zp(num,den)	Función de Transfer. a cero-polo
zp2ss	[A,B,C,D]=zp2ss(Z,P,k)	Cero-polo a Espacio de Estados
zp2tf	[num,den]=zp2tf(Z,P,k)	Cero-polo a Función de Transferencia

MANIPULACIÓN DE SISTEMAS LINEALES
Combinación de Sistemas: Regulación Automática. Consultar Manual.

ANÁLISIS DE SISTEMAS LINEALES
Regulación Automática Incluye Lugar de las Raíces). Consultar Manual.

DISEÑO DE SISTEMAS LINEALES
Consultar Manual.

RESPUESTA EN EL DOMINIO TEMPORAL		
dimpulse	dimpulse(A,B,C,D,iu) dimpulse(A,B,C,D,iu,n) dimpulse(num,den) dimpulse(num,den,n)	Respuesta a un impulso unidad discreto
dstep	dstep(A,B,C,D,iu) dstep(A,B,C,D,iu,n) dstep(num,den) dstep(num,den,n)	Respuesta a un salto unidad discreto
filter	Y=filter(num,den,u) [y,Zf]=filter(num,den,u,Zi)	Respuesta de un filtro discreto
impulse	impulse(A,B,C,D,iu) impulse(A,B,C,D,iu,t) impulse(num,den) impulse(num,den,t)	Respuesta a un impulso unidad continuo
lsim	lsim(A,B,C,D,U,t) lsim(A,B,C,D,U,t,Xo) lsim(num,den,u,t)	Respuesta de un sistema lineal continuo a una entrada arbitraria
step	step(A,B,C,D,iu) step(A,B,C,D,iu,t) step(num,den) step(num,den,t)	Respuesta a un salto unidad continuo

MISCELANEA
Consultar Manual.

III.3.-EJEMPLOS

Vamos a ver algunos ejemplos de las partes de más interés, concretamente veremos solo sistemas continuos (respuesta en frecuencia y temporal), un ejemplo de filtrado y otro de análisis espectral.

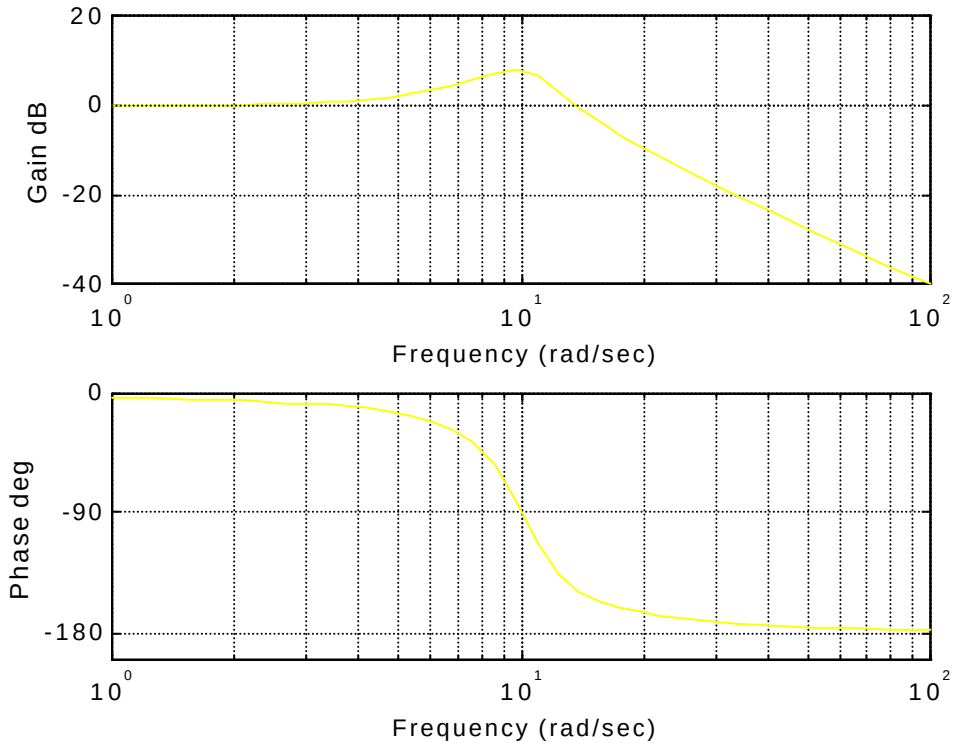
III.3.1.-SISTEMAS CONTINUOS.

Los sistemas continuos son sistemas cuyas dinámicas y señales varían continuamente con el tiempo. Ejemplos típicos incluyen circuitos eléctricos. Para el siguiente ejemplo usaremos la función de transferencia:

$$H(s) = \frac{100}{s^2 + 4s + 100}$$

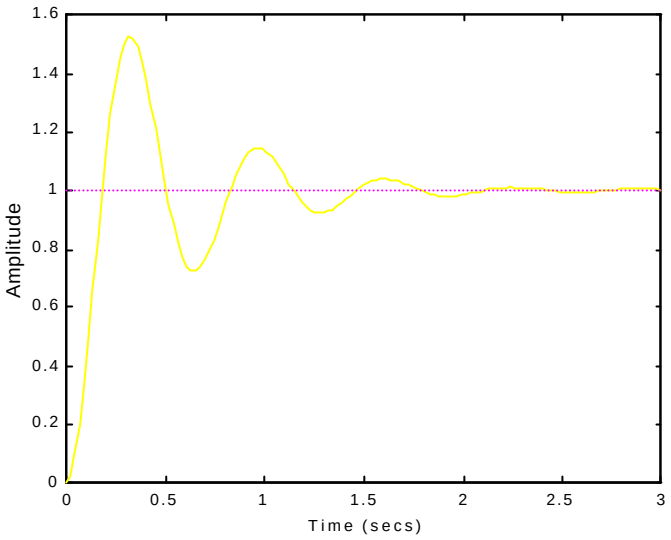
La respuesta en frecuencia de este sistema se puede dibujar usando la función de Bode:

```
>>nc=100;dc=[1 4 100];
>>bode(nc,dc)
```



Su respuesta a un escalón unitario se puede dibujar (véase siguiente gráfica) con la función sep:

```
>>step(nc,dc)
```



Una descripción en el espacio de estados de este sistema viene dada por:

```
>>[A,B,C,D]=tf2ss(nc,dc)
A=
        -4   -100
         1      0
B=
        1
        0
C=
        0   100
D=
        0
```

Finalmente, convertir este sistema a un sistema discreto equivalente usando un retenedor de orden cero con un periodo de muestreo de 0.01 segundos da:

```
>>[n,d]=c2d(nc,dc,0.01)
n=
        2.7183
d=
        0.0172          0.0687          1.7183
```

III.3.2.-FILTRADO.

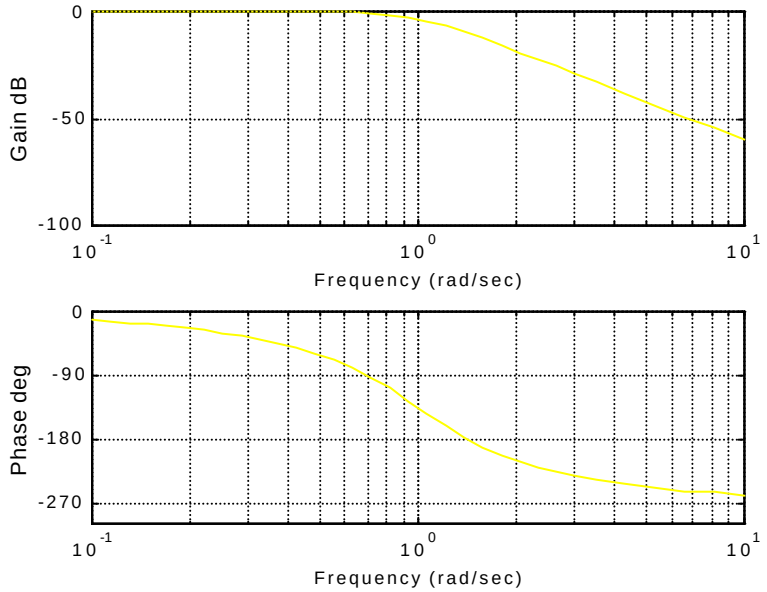
Un filtro de Butterworth pasabaja analógico de n-ésimo orden, con ancho de banda de 3db se obtiene usando **buttap(n)**. Por ejemplo, el filtro de tercer orden está dado por:

```
>>[Z,P,K]=buttap(3)

Z=
        []
```

```
P=
        -0.5000 + 0.8660i
        -1.0000 + 0.0000i
        -0.5000 – 0.8660i
K=
        1

>>[n,d]=zp2tf(Z,P,K)  %convierte cero-polo a notación de función de Transf.
n=
        0       0       0       1
d=
        1.0000          2.0000          2.0000          1.000
>>bode(n,d)     %calcula la respuesta en frecuencia
```



III.3.3.-ANÁLISIS ESPECTRAL.

El algoritmo FFT encuentra gran aplicación en el procesamiento de señales digitales. Su uso incluye filtrado, convolución, cálculo de la respuesta en frecuencia y de retardo de grupo y estimación espectral de potencia.

En este ejemplo vamos a estimar la transformada de Fourier continua de la señal

$$f(t)=\begin{cases}12e^{-3t} & ;\ t\geq 0\\ 0 & ;\ t<0\end{cases}$$

Analíticamente, esta transformada de Fourier está dada por

$$F(w)=\frac{12}{3+jw}$$

Aunque usar en este caso la FFT tiene poco valor, ya que se conoce la solución analítica, este ejemplo ilustra un método para estimar la Transformada de

Fourier de señales menos comunes. Las siguientes órdenes de MATLAB estiman |F(w)| y la compara gráficamente con la expresión analítica:

>>N=128;

hace que el número de puntos sea una potencia de 2, lo cual acelera el cálculo de la FFT,

>>t=lispace(0,3,N);

>>f=2*exp(-3*t)

identifica los puntos de muestreo y evalúa f(t) en estos puntos. En el punto de muestreo final la función f(t) es aproximadamente cero para minimizar el "aliasing".

>>Ts=t(2)-t(1);

>>Ws=2*pi/Ts;

identifica el periodo de muestreo y la frecuencia de muestreo en rd/s.

>>F=fft(f);

obtiene la FFT de f(t),

>>Fp=F(1:N/2+1)*Ts;

extrae sólo las componentes de frecuencia positiva de F y las multiplica por el periodo de muestreo para estimas F(w).

>>W=Ws*(0:N/2)/N;

crea el eje de frecuencia continuo, el cual comienza en cero y termina en la frecuencia de Nyquist Ws/2,

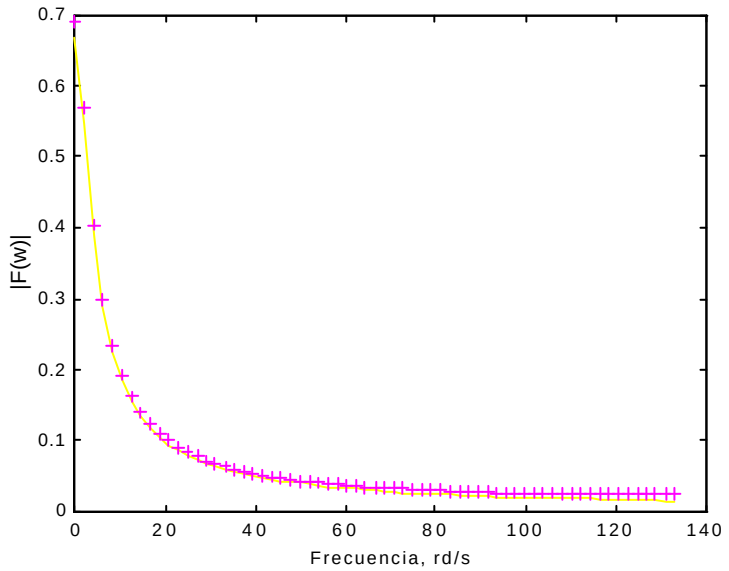
>>Fa=2./(3+j*W);

evalúa la transformada de Fourier analítica,

>>plot(W,abs(Fa),abs(Fp),'+')

>>xlabel('Frecuencia, rd/s'),ylabel('|F(w)|')

genera el siguiente dibujo donde la línea continua es la solución analítica y las marcas + son las estimaciones de FFT.



Ejercicios de

MATLAB

1.-Matlab Básico

2.-Toolbox de Matemática Simbólica

Ejercicios de Matlab

MATLAB BÁSICO

1º.-Crear un vector de valores comprendidos entre 5 y 500, con 200 valores uniformemente distanciados.

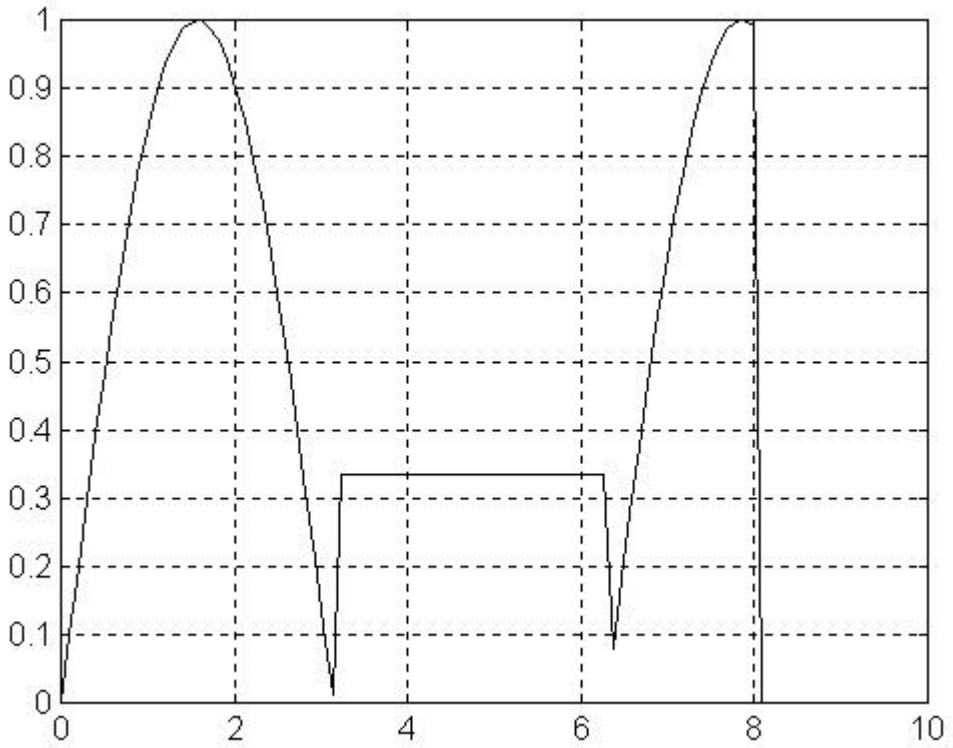
2º.-Generar un vector de valores comprendidos entre 1 y 50, con incrementos uniformes de 0.02.

3º.-Generar un vector de valores comprendidos entre 10 y 10000, con 4 valores, espaciados logarítmicamente.

4º.-Generar un conjunto de 50 valores, repartidos uniformemente entre 0 y 2π.

5º.-Dividir el intervalo $[0,\pi]$ en tramos de 0.01rd de anchura.

6º.-Definir una función z tal que, dibujada, sea: sen(x) cuando ésta sea positiva; 1/3 cuando sin(x) sea negativa, y 0 cuando x>8.



7º.-Resolver en forma matricial ([A]· [X]=[B]) el siguiente sistema de ecuaciones, de dos formas distintas:

$$\begin{bmatrix} 1 & 3 & -5 \\ -1 & 1 & 3 \\ 2 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \\ -10 \end{bmatrix}$$

8º.-Dada la matriz anterior [A], obtener:

a) Rango de dicha matriz.
b) Determinante.
c) Autovalores (raíces del polinomio característico) y autovectores.
d) Polinomio característico.

9º.-Escribir las diez primeras potencias de 2, usando un bucle **FOR**.

10º.-Escribir las potencias enteras de 2, mientras éstas sean inferiores, en valor, a 5000. (Usar un bucle **WHILE**).

11º.-Construir la función primo(n) que nos diga si el número n es o no primo (en caso de no serlo, debe indicar cual es su primer divisor).

12º.-Disponemos de la siguiente tabla, donde se muestran las temperaturas máximas durante la última semana en Madrid, Barcelona y Valencia.:

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
Madrid	13	15	14	12	13	11	14
Barcelona	15	16	17	13	15	12	14
Valencia	17	19	21	18	19	20	19

(nota, aunque se escriba de esa forma, las variables –temperaturas- deben ponerse como columnas)
Obtener:

a)Temperatura media de cada ciudad, y media total entre las tres.
b)Temperatura máxima y mínima en cada ciudad.
c)Temperatura mínima de cada ciudad y el día en que se produjo.
d)Desviación estándar.
e)Diferencia de temperatura entre cada día y valor medio de estas variaciones.
f)Ordenar las temperaturas (junto con los días producidos).
g)Dibujar una gráfica que contenga todas esas temperaturas.

13º.-¿Cómo construirías la operación n! (factorial de n) usando la función **prod**?.

14º.-Dado el polinomio $P(x)= x^5 – 10x^4 + 40x^3 – 80x^2 + 79x – 30$:

a) Obtener sus raíces.
b) Multiplicarlo por $x^2 + 1$
c) Dividirlo por $x^2 + x + 1$
d) Derivarlo
e) Sumarle $x^2 + 1$
f) Evaluarlo en los puntos x=-1, x=0 y x=1.
g) Dibujarlo entre [0.5, 3.5] con 100 puntos.

15º.-Encontrar un polinomio cuyas raíces sean -1, 1, 2 y $\pm i$

16º.-Desarrollar en fracciones simples la expresión

$$\frac{x+5}{x^4-5x^3+5x^2+5x-6}$$

17º.-Un cuerpo se deja caer desde una cierta posición inicial x_0 con una velocidad inicial v_0. Los datos experimentales obtenidos son los siguientes:

t(s)	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0
x(m)	5.1	7.3	11.9	19.0	28.5	40.5	54.9	71.8	91.1

Encontrar los valores de g, v_0 y x_0 y dibujar los puntos experimentales (marcándolos con o) y la curva aproximadora más idónea, por mínimos cuadrados. Como es obvio, se cumple:

$$x(t)=\frac{1}{2}g\cdot t^2+v_o\cdot t+x_o$$

18º.-La temperatura registrada en una ciudad durante un día de verano, tomada cada hora es la siguiente:

t(h.)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T(ºC)	14	13	12	12	10	9	6	6	7	7	9	12	13	15	17	19	20	21	20	20	18	16	14	13

¿cuál sería la temperatura esperada a las 16'5 horas:

a) Por interpolación lineal.
b) Por interpolación mediante "splines".

Dibujar los puntos experimentales y la interpolación mediante "splines"

19º.-Dada la función $f(x) = e^{-x}\cdot \cos(x)$:

a) Dibujarla entre 0 y 2π.
b) Obtener el primer mínimo.
c) Obtener la integración numérica entre 2 y 3, por los métodos trapezoidal y cuadrático.

20º.-Dibujar las funciones $\text{sen}^2(x)$ y cos(2x) entre $[0,2\pi]$, usando 50 puntos.

a) Añadir rejilla.
b) Titular la gráfica como "funciones trigonométricas".
c) Dibujar $\text{sen}^2(x)$ en rojo, línea punteada y marcar con "x".
d) Dibujar cos(2x) en azul, a trazos y marcar con "o" (conjuntamente con la anterior).
e) Poner un título a cada eje.
f) Señalar cada curva con un título dentro del gráfico
    a. Con la orden **text**.
    b. Con la orden **gtext**.
g) Sobre esa misma gráfica, dibujar sen(2x) en verde y trazo continuo.

21º.-Dibujar las dos primeras curvas anteriores simultáneamente, pero en dos "plots", uno al lado del otro.

22º.-Dibujar la cardioide r = 2· (1+cost). Tomar 10 valores de la curva usando **ginput**.

23º.-Las notas obtenidas en un examen, y el número de alumnos que han obtenido cada una, son las siguientes:

Nota	0	1	2	3	4	5	6	7	8	9	10
Nº al.	1	3	4	4	9	12	5	2	3	1	0

a) Dibujar un diagrama de barras con dicha información.
b) Dibujar un diagrama de escalera.

24º.-La relación de notas obtenidas en otro examen es:

1, 2, 2, 7, 4, 5, 3, 2, 1, 4, 5, 6, 6, 5, 5, 4, 3, 1

Dibujar un histograma que contenga dicha información.

25º.-Generar aleatoriamente (con distribución normal) 1000 números y obtener su histograma entre –2 y 2, con incrementos de 0.1

26º.-Generar aleatoriamente (con distribución normal) 50 números y dibujar dichos puntos, unidos con una línea a trazos vertical al eje de abcisas.

27º.-Los datos experimentales obtenidos en una práctica son los siguientes:

x	0	0.5	1.0	1.5	2.1	2.7	3.4	4.1	4.8	5.5	6.3
y	1.0	1.22	1.39	1.53	1.64	1.72	1.82	1.89	1.91	1.94	1.96
error	0.05	0.05	0.05	0.05	0.05	0.1	0.1	0.1	0.1	0.1	0.1

Representar esos datos con sus intervalos de error.

28º.-Dibujar la función sen(x)/x entre –15 y 15, usando la orden **fplot**.

29º.-Dibujar en 3-D la hélice siguiente: f(t) = (sen(t),cos(t),t)
(nota: valuar t entre 0 y 10π, con incrementos de π/50).

30º.-Dibujar la función $Z = \frac{sen\left(\sqrt{x^2 + y^2}\right)}{\sqrt{x^2 + y^2}}$ en la malla x $\in$ (-7.5,7.5) ; y $\in$ (-7.5,7.5).

(nota: cuidado con posibles 0/0):

a) Gráfica de malla.
b) Gráfica de superficie.
c) Gráfica de contorno 2-D con diez contornos.
d) Gráfica de contorno 3-D con diez contornos.
e) Gráfica de color.
f) Simultanear c) y f) (eliminar la rejilla con “shading flat”)

Ejercicios de Matlab

Toolbox de MATEMÁTICA SIMBÓLICA

1º.-¿cuál es el resultado de la operación 2 + symop('(2*4+1)/3-1')?. ¿Por qué?.

2º.-¿Cuál es la variable independiente de las expresiones siguientes?:

a) a*s2+3v
b) var + 5
c) sin(3j+1)
d) cos((2s+3t)/(u+2))
e) symvar('cos((2s+3t)/(u+2))','r')

3º.-Extraer numerador y denominador de:

$$\begin{bmatrix} \frac{3}{2}+\frac{1}{5} & \frac{3x^2+1}{2x} \\ x+\frac{3x^2}{2} & \frac{2a}{(x+2)(x+1)} \end{bmatrix}$$

4º.-Dadas las funciones $f = 3x^2 - 2x + 1$; $g = x^2 + 3x - 1$ obtener:

a) f+g
b) f-g
c) f*g
d) f/g
e) f^{2x}
f) (2f-3)/g+2 con una sola orden.

5º.-Dadas las siguientes funciones

$$f = \frac{x}{x^2+4} \qquad g = \cos(2x) \qquad h = \frac{u}{u^2+4} \qquad k = \cos(2v)$$

obtener:

a) f(g(x))
b) g(f(x))
c) h(k(v))
d) $g^{-1}(x)$

6º.-Obtener la inversa de $f(t) = \frac{2}{5x+t}$

7º.-Obtener las siguientes sumas:

a) $\sum_{1}^{3} x^2$

b) $\sum_{1}^{n} (2n-1)^2$

c) $\sum_{1}^{\infty} \frac{1}{(2n-1)^2}$

8º.-Dado el polinomio P = [1 2 3 4], pasarlo a simbólico, con la variable s.

9º.-Dada la función $f(x) = x^2 - 2x + 3$:

a) Sustituir la variable x por s.
b) Sustituir la variable x por 2 y pasar a numérico.

10º.-Dada la función $f = x^3 + 2a^2x^2 - 3x + 1$ obtener:

a) df/dx
b) d^2f/dx^2
c) d^2f/da^2
d) $\int f dx$
e) $\int f da$
f) $\int_0^2 f dx$
g) $\int_0^2 f da$
h) $\int_m^n f dx$
i) $\int_m^n f da$

11º.-Dada la función $f = -4.5x^2 + 2x + 100$:

a) Representarla con **ezplot**.
b) Corregir, para representar solamente valores de x > -1.

12º.-Dada la función $f(x) = ((x+1)(x+2)-(x+2)^2)^3$:

a) Expandirla.
b) Factorizarla.
c) Visualizar su expresión de la forma matemática habitual.

13º)Simplificar la expresión: $\frac{tg(x)}{tg(2x)-tg(x)}$

(pruébese la función **simplify** y la función **simple** repetida dos veces)

14º.-Expandir en fracciones la expresión: $\frac{2x^3+14x^2+33x+23}{x^3+6x^2+11x+6}$

15º.-Obtener la precisión actual.

16º.-Cambiar la precisión actual a 18 dígitos.

17º.-Obtener el valor de π con esos 18 dígitos.

18º.-Obtener el valor de π con 200 dígitos de precisión (sin variar la precisión total de MATLAB).

19º.-Obtener la expresión general de la solución de una ecuación de 2º orden de la forma típica: $ax^2 + bx + c$.

20º.-Resolver las siguientes ecuaciones algebraicas:

a) $x^3 - 7x^2 + 3ax = 0$
b) $x^5 + 2x + 1 = 0$
c) $\ln(x) = e^{-x}$
d) $\text{sen}(x) = tg(2x)$

21º.-Resolver el siguiente sistema de ecuaciones:

$$\left.\begin{array}{c} 3x-2y+z=-3 \\ -x+y-3z=1 \\ x+y+z=0 \end{array}\right\}$$

22º.-Resolver las siguientes ecuaciones diferenciales:

a) $y'' - 3y' + 2y = 2e^x(1-x)$
b) $(x-1)y'' - xy' + y = 0$
c) $x^3y''' + 2xy' - 2y = x^2\ln(x) + 3x$
d) $y'' + 3y' - 2y = e^{-x}\cos(x)$ con $y(0)=1$; $y'(0) = -1$

23º.-Resolver el siguiente sistema de ecuaciones diferenciales:

a)

$$\left.\begin{array}{c} 2\frac{dx}{dt}+\frac{dy}{dt}-4x-y=e^t \\ \frac{dx}{dt}+3x+y=0 \end{array}\right\}$$

b)

$$\left.\begin{array}{l} \dfrac{dx}{dt}+\dfrac{dy}{dt}+y=1 \\ \dfrac{dx}{dt}-\dfrac{dz}{dt}+2x+z=1 \\ \dfrac{dy}{dt}+\dfrac{dz}{dt}+y+2z=0 \end{array}\right\}$$

c)

$$\left.\begin{array}{l} \dfrac{d^2x}{dt^2}-2x-3y=e^{2t} \\ \dfrac{d^2y}{d^2}+2y+x=0 \end{array}\right\}$$

24º.-Dada la matriz simbólica:

$$M=\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

a) Obtener la dimensión de M.
b) Encontrar el elemento de la segunda fila, tercera columna de la matriz.
c) Sustituir el elemento 'i' por 'k'.
d) Sumar x a todos los elementos de M.

25º.-Dada la matriz simbólica:

$$M=\begin{bmatrix} sen(x) & \cos(x) \\ -\cos(x) & sen(x) \end{bmatrix}$$

Obtener:

a) M*M
b) M^3
c) Simplificar el resultado anterior.
d) Traspuesta.
e) Determinante.
f) Polinomio característico.
g) Autovalores y autovectores

26º.-Resolver, simbólicamente, usando la orden **linsolve**, la ecuación A· X = B, siendo

$$A=\begin{bmatrix} 1 & 2 & 1 & -1 \\ 2 & 2 & 1 & 1 \\ 1 & 3 & 1 & 0 \\ -1 & -1 & 1 & 2 \end{bmatrix} \qquad B=\begin{bmatrix} 0 \\ 1 \\ 2 \\ 0 \end{bmatrix}$$

27º.-Obtener la transformada de Laplace de la función $f(t) = e^{-2t} \cos(3t) + 2t^2 + 1$

28º.-Obtener la transformada inversa de Laplace de la función

$$F(s) = \frac{2s^4 + 6s^3 + 17s^2 + 16s + 52}{s^3\left(s^2 + 4s + 13\right)}$$

29º.-Obtener la transformada de Fourier de $f(t) = e^{-3t} \cdot u(t)$ (siendo u(t) la función escalón unitario).

30º.-Obtener la transformada inversa de Fourier de:

$$F(w) = \frac{1}{3 + i \cdot w}$$

31º.-Probar las posibilidades de la herramienta **funtool**.