Práctica Nº1
Introducción a MatLab

1.- Introducción.

MatLab[1] es un sistema de programación y cálculo basado en la manipulación de matrices. El nombre mismo del sistema o paquete de cómputo proviene de la abreviación **Mat**riz **Lab**oratory o Laboratorio de Matrices. Por lo tanto, la filosofía detrás de la computación por medio de este sistema es considerar a los objetos (tanto matemático como gráficos) como matrices de forma que podamos usar el álgebra matricial y otras propiedades para ahorrar tiempo de cómputo. Las características principales es que es un lenguaje de alto nivel para el cálculo numérico, visualización y desarrollo de aplicaciones; para ello cuenta con funciones matemáticas para álgebra lineal, estadística, análisis de Fourier, optimización integración numérica y resolución de ecuaciones diferenciales ordinarias. Tiene gráficos integrados para la visualización de datos y herramientas para la creación de diagramas personalizado, además cuenta con herramientas de desarrollo para mejorar la calidad del código, facilidad de mantenimiento y maximizar el rendimiento. Presenta herramientas para la creación de aplicaciones con interfaces gráficas personalizadas y tiene funciones para integrar los algoritmos basados en MATLAB con aplicaciones externas: lenguajes (C, Java, etc.) y programas (Microsoft Excel). MATLAB es una potente herramienta de cálculo numérico y simbólico, análisis de datos, simulación y visualización gráfica. Es de uso muy difundido en ámbitos académicos, científicos y técnicos para el desarrollo de tareas de investigación. Tiene la gran ventaja de ser un lenguaje de alto nivel que integra, en un único ambiente, rutinas de cálculo, visualización, simulación y programación. Este software es de amplio uso ya que los problemas se pueden formular utilizando una notación matemática estándar. La representación básica de los datos en **MATLAB** es en forma matricial. Así, por ejemplo, un número cualquiera (un escalar) es considerado como una matriz 1×1, o un vector como una matriz de una sola fila o columna.

Se puede trabajar con MATLAB directamente por medio de la invocación de comandos en la ventana de trabajo (Command Window) que es la ventana en donde se escriben los comandos o bien a través de procesos estructurados (rutinas o funciones con la extensión ***.m**).

Algunos de los usos más comunes de **MATLAB** son, por ejemplo:

- Cálculo numérico.
- Desarrollo de algoritmos.
- Modelado, simulación y desarrollo de prototipos.
- Análisis y visualización de datos.
- Construcción de gráficas.

MATLAB es un sistema abierto al cual el usuario puede incorporar nuevas funciones y programas para su utilización en aplicaciones particulares. **MATLAB** dispone de un código básico y de varias librerías especializadas denominadas **Toolboxes**, que permiten resolver problemas específicos en diversas áreas de ciencia

[1] MatLab, creado por Cleve Moler en 1984 y es propiedad de The Math Works, Inc.

e ingeniería. MATLAB incluye una gran cantidad de funciones predefinidas, que ayudan a realizar cálculos de todo tipo, así como para visualización de datos y resultados. Actualmente existen Toolboxes en áreas tales como Control, Procesamiento de Señales, Identificación, Procesamiento de Imágenes, Redes Neuronales, Wavelets, etc. Probablemente la característica más importante de MATLAB es su capacidad de crecimiento. Esto permite convertir al usuario en un autor contribuyente, creando sus propias aplicaciones. En resumen, las prestaciones más importantes de MATLAB son:

- Escritura del programa en lenguaje matemático.
- Implementación de las matrices como elemento básico del lenguaje, lo que permite una gran reducción del código, al no necesitar implementar el cálculo matricial.
- Implementación de aritmética compleja.
- Un gran contenido de órdenes específicas, agrupadas en TOOLBOXES.
- Posibilidad de ampliar y adaptar el lenguaje, mediante archivos (script y funciones).

En esta primera práctica, esperamos que el alumno se familiarice con los comandos básicos de **MATLAB** de forma de poder realizar el tratamiento de los datos obtenidos en las prácticas siguientes.
Recomendamos abrir MATLAB en otra ventana e ir practicando los distintos comandos a medida que se leen estas notas.

2.- Iniciando una sesión de MatLab.

MATLAB se puede iniciar como cualquier otra aplicación de Windows, haciendo doble clic en el icono correspondiente en el escritorio o por medio del menú inicio tecleando la palabra MatLab. Al iniciar, se abre una ventana similar a la que se muestra en la Figura 1, es la vista por defecto con las que cuenta MatLab.

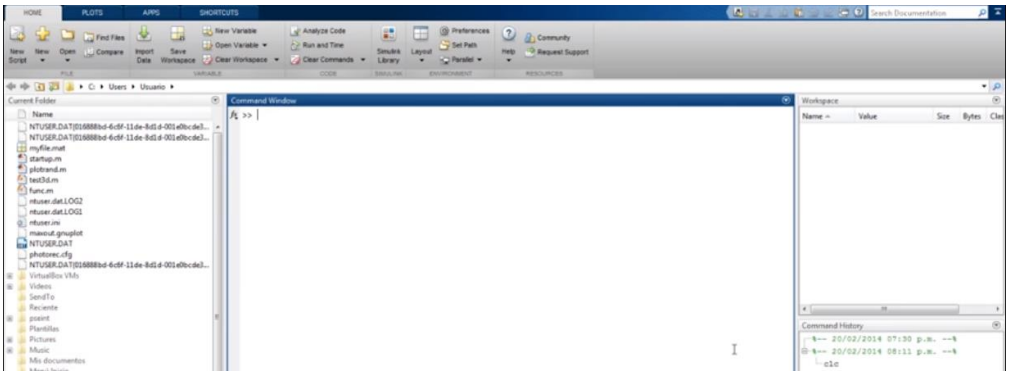

Figura 1 Menú Inicio de MATLAB por defecto

Esta configuración puede ser cambiada fácilmente por el usuario, en caso que aparezca diferente, una visita similar se puede conseguir haciendo clic en la opción ***Layout*** y ***Default*** de la sección **ENVIRONMENT** (Figura 2)

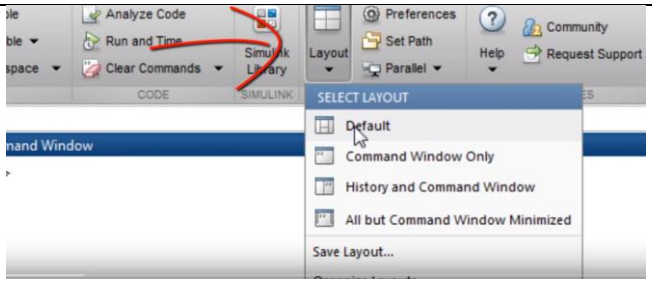

Figura 2. Recuperación del menú inicio Default del MatLab. La flecha roja indica la opción de selección del menú inicio

2.1.- El entorno

El espacio de trabajo de MATLAB es muy gráfico e intuitivo similar al de otras aplicaciones profesionales de windows, las componentes más importantes del entorno de trabajo de son las siguientes: MATLAB Desktop (escritorio de MatLab), Command Window, Command History, Workspace, Editor & Debugger, Array Editor y Help.

MATLAB Desktop (escritorio de MatLab) es la ventana más general de la aplicación, el resto de las ventanas o componentes pueden alojarse en el escritorio MATLAB o ejecutarse como ventanas independientes. Los componentes del escritorio MATLAB pueden aparecer como subventanas independientes o como pestañas dentro de una de las subventanas. Ello ofrece una gran flexibilidad, permitiendo a cada usuario decidir la forma de utilizar; desde la opción Layout es posible controlar las componentes visibles y la forma en que se visualiza (Figura 3) Tiene diferentes entornos de trabajo predefinidos y se puede seleccionar cuál de estas subventanas queremos que se muestren o no.

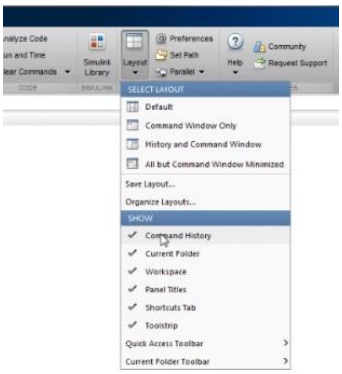

Figura 3. Selección de ventanas a visualizar

2.1.1.- Command Window - (Ventana de Comandos)

Esta ventana es la más importante del programa. Común a todas las versiones del MATLAB es donde se realizan las operaciones. Es donde se ejecutan interactivamente las instrucciones de MATLAB y es donde se muestran los resultados correspondientes si es el caso en cierta forma es la ventana más importante. Aparece un **prompt** o (aviso) >>, indicando que el programa está preparado para recibir instrucciones para apreciar el uso de ésta su ventana ingresemos la siguiente línea *a* igual a *3* y pulsar la tecla ***Enter*** la variable *a* se guarda con el valor de 3 y el resultado se muestra en pantalla. Debemos pensar que MATLAB está creado para realizar tareas y guardar los resultados o programas que nos llevan a ellos. No se trata de un editor donde ir escribiendo el trabajo realizado.

Por ello en esta pantalla no se nos permitirá volver atrás y rectificar. Todo lo realizado va quedando en ella hasta que decidamos borrarlo.

Existen maneras de recuperar lo escrito para no tener que volver a escribirlo si deseamos corregir o volver a ejecutar:

- Puede hacerse con las flechas del teclado:
  Con ellas se recupera lo escrito con anterioridad en la línea del >> pudiéndose corregir y volver a ejecutar.
- Pinchando dos veces en la línea correspondiente del Command History.
  Esto permite la ejecución inmediata de lo elegido.
- Pinchando con el botón derecho del ratón sobre la sentencia ejecutada en el Command Window. Esto permite volver a ejecutar dicha operación, copiarla, etc.

2.1.2.- Command History – (Historial de Comandos)

Todos los comandos que hayan sido ingresados al *Command Window* son guardados en esta subventana, inclusive se guardan los comandos usados en sesiones pasadas, permitiendo tener una visión más general de lo hecho anteriormente y ejecutar algunos de los comandos que hay en ella simplemente haciendo doble clic con el mouse.

2.1.3.- Workspace (Espacio de trabajo)

Esta ventana del espacio de trabajo permite la visualización de las variables almacenadas en memoria (Figura 4). Da una información completa sobre ellas: nombre, tipo, tamaño y clase. También se visualiza las funciones creadas por el usuario. Todas las variables que se muestran en el workspace son variables del espacio de trabajo base. Se verá que cada función tiene su propio espacio de trabajo con variables cuyos nombres no interfieren con las variables de los otros espacios de trabajo.

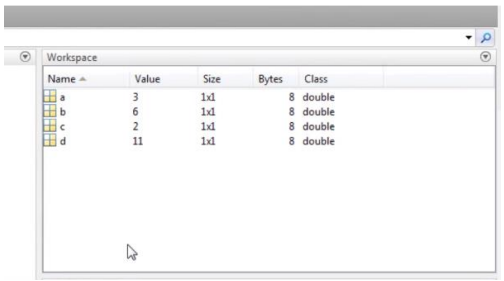

Figura 4. Subventana Workspace con variables pre creadas.

A modo de ejemplo vemos las cuatro variables creadas anteriormente (a, b, c, d) y su respectivo valor, tamaño, y clase; ya que MATLAB trata todas las variables como *arrays*, el tamaño se representa como un array de 1×1, la clase y los bytes que ocupan las variables (se verán más adelante). Es posible cambiar el nombre de las variables en cualquier momento usando el *workspace*, se selecciona la variable a la que deseamos cambiar el nombre y le damos clic o simplemente dando clic derecho y seleccionando la opción *rename* y designamos el nuevo nombre. (existe otras de las posibilidades a las que podemos acceder usando clic derecho)

2.1.4.- Array Editor

Al hacer doble clic sobre una variable en el workshop se abre una nueva ventana llamada Array editor (Figura 5), la cual permite ver y modificar los valores de los elementos de cualquier matriz o vector haciendo clic sobre la celda correspondiente. Los nuevos valores (y/o tamaño) se visualizarán en el Workspace.

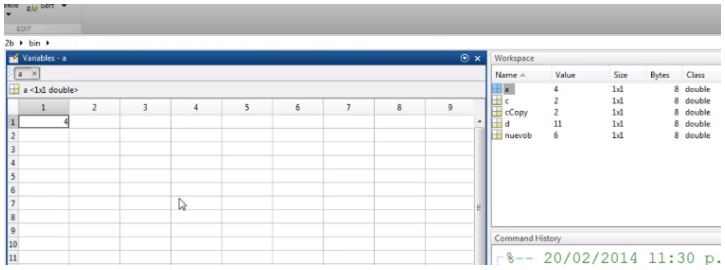

Figura 5. Array Editor, visualizando la variable a.

2.1.5.- Current Folder – (Carpeta Actual)

Current Folder (Carpeta Actual o directorio activo) es muy importante en MatLab, los programas de MATLAB se encuentran en archivos con la extensión punto m (*.m*), estos archivos ejecutan tecleando su nombre en la línea de comandos seguido de los argumentos entre paréntesis (si se trata de funciones)

Para que un archivo *.m* se pueda ejecutar es necesario que se cumpla una de las siguientes dos condiciones:

- *que esté en el directorio actual*- MATLAB mantiene en todo momento un único directorio con esta condición, este directorio es el primer sitio en el que MATLAB busca cuando desde la línea de comandos se le pide que se ejecute un archivo,
- *que esté en uno de los directorios indicados en el Path de MatLab*- el *path* (ruta de directorio) es una lista ordenada de directorios en los que el programa busca los archivos con las funciones a ejecutar.

Muchos de los directorios del *path* son propios de MatLab, pero los usuarios también pueden añadir sus propios directorios normalmente al principio o al final de la lista.

La subventana Current Folder permite explorar los directorios del ordenador en forma análoga a la del explorador u otras aplicaciones de Windows. Cuando se llega al directorio deseado se muestran los archivos allí contenidos permitiendo, entre otras cosas, ordenarlos (por fecha, tamaño, nombre, etc.)

El directorio actual cambia automáticamente en función del directorio seleccionado, se puede cambiar desde la propia barra de herramientas del MATLAB (Figura 6).

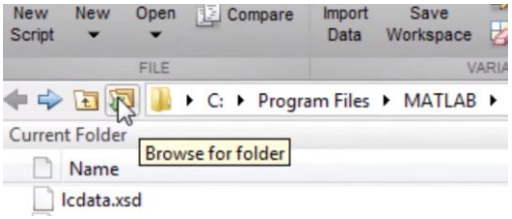

Figura 6. Botón explorador para seleccionar directorios y carpetas

2.1.6.- Editor/Debugger

El editor es la ventana en la cual se puede escribir, editar, crear y guardar sus propios programas en archivos llamados ***m-files*** o archivos ***m***

Para acceder al editor basta con hacer clic en el botón New Script (Figura 7) de la parte superior izquierda Estos archivos ***m-files*** (de extensión ***.m***) contienen conjuntos de comandos o definición de funciones.

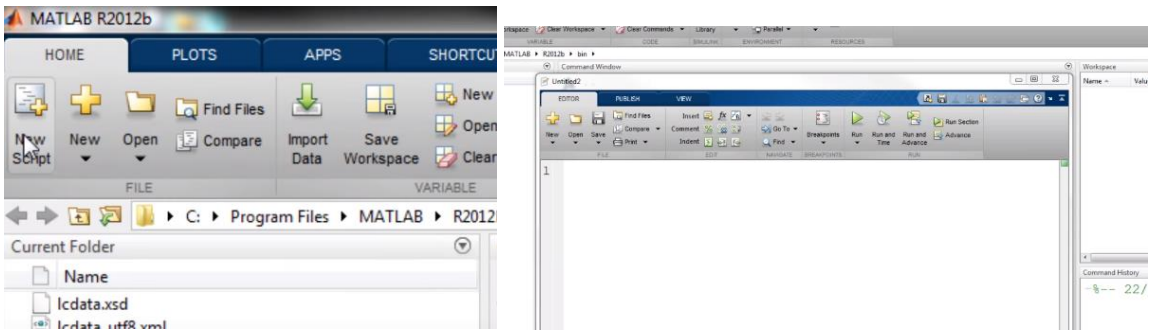

Figura 7. Botón New Script (izquierda) para crear un nuevo editor (derecha)

Al teclear su nombre en la línea de comandos y pulsar *Enter* se ejecutan uno tras otro todos los comandos contenidos en dicho archivo; el poder guardar instrucciones y grandes matrices en un archivo permite ahorrar mucho trabajo de tecleado. Este tipo de archivos se pueden crear con cualquier editor de texto (ejemplo el bloc de notas) pero MATLAB dispone de un editor que permite crear, modificar y ejecutarlos paso a paso para ver si contienen errores; dicho proceso se conoce como *Debugger* o depuración

Veamos un archivo (previamente creado) llamado *ejemplo2.m* se pueden ver los comentarios (primera y octava línea) y al final (línea 14) una línea de mensaje en pantalla *Ya he terminado*. El editor muestra con diferentes colores los diferentes tipos de comandos: en verde los comentarios, en violeta las cadenas de caracteres, etc. (Figura 8)

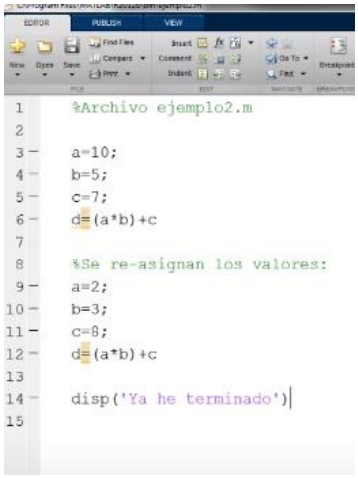

Figura 8 Algunos comandos en el editor

En el caso que se tenga un error (ejemplo no se cierra un paréntesis, como ser en la línea 14 se escribe ***disp('Ya he terminado'***) en la parte derecha del editor se muestra una pequeña barra de color rojo indicando que en esa línea hay un error y si nos paramos con el cursor sobre ella MATLAB muestra una descripción del error.

2.1.6.1 Comentarios[2]

Una de las herramientas más útiles para hacer más legible y mejor estructurado un código fuente son los comentarios, estos describen el código permitiendo que otras personas lo entiendan y además de ayudar a recordar cuando éstos se necesiten más tarde.

En MatLab, los comentarios se introducen con el símbolo porcentaje (%), las líneas 1 y 8 de la Figura 8 son ejemplos de comentarios. Cuando el símbolo porcentaje aparece, todo lo que le sigue a partir de él hasta el final de esa línea se considera como un comentario y simplemente es ignorado por MatLab.

Esencialmente existen dos tipos de comentarios: para una sola línea y para varias líneas (se abre con **%{** y se cierra con **%}**, todo lo que se escriba en el medio serán comentarios)

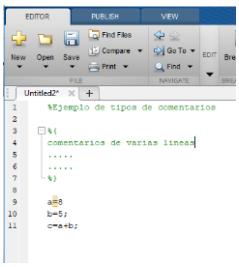

Figura 9 Diferentes tipos de comentarios

Los comentarios pueden encontrarse en diferentes lugares del código: al principio, se conoce como *general comment* o *comentario general* (por lo general se incluye toda la información sobre quién lo escribió, la fecha de creación y el objetivo del código); arriba de cada función, se conoce como *función header* o *encabezado de función* (proporciona información sobre el propósito de una determinada función); en alguna línea intermedia del código, se le llama *in line comment* o *comentario en la línea* (explica lo que hace una determinada línea del código y se emplea con frecuencia en aquellas sentencias complicadas).

2.1.7.- Help

MATLAB dispone de una excelente ayuda con la que es posible encontrar la información que se desee. Se puede acceder haciendo clic en el botón *Help* de la ventana principal de la aplicación o tipeando ***help*** en la ventana de comando. Si en particular se busca la ayuda sobre una función en particular, como ser *sum* (suma), debe escribirse: ***help sum***. Esto significa que debemos saber cómo se llama la función. Si introducimos *help* sin ningún argumento nos aparecerá una ayuda general desde donde podremos encontrar cualquiera de las funciones disponibles.

2.2.- Directorio de trabajo

El programa **MATLAB** se inicia por defecto en el directorio indicado como "Current Folder". Antes de comenzar a trabajar es conveniente cambiarse al directorio de trabajo. Para cambiar de directorio se pueden utilizar las

[2] La utilidad de los mismos se verán más adelante

herramientas de desplazamiento que aparecen en la dicha ventana (rectángulo rojo en la Figura 10) o el comando *cd* (**c**hange **d**irectory) como se indica: **>> cd E:\users**

Para crear una nueva carpeta se utilizan las herramientas encerradas en el Rectángulo azul (ver Figura 10), entre otras opciones se puede emplear la siguiente sintaxis desde la command window: ***>> mkdir('newdir').*** La nueva carpeta también puede ser creada de forma usual en el directorio que se trabaja. De esta manera se crea una nueva carpeta llamada "*newdir*" en la carpeta en que estaba trabajando. Para verificar en cual directorio se está trabajando se utiliza el comando: *pwd* (***p**ath of **w**orking **d**irectory*).

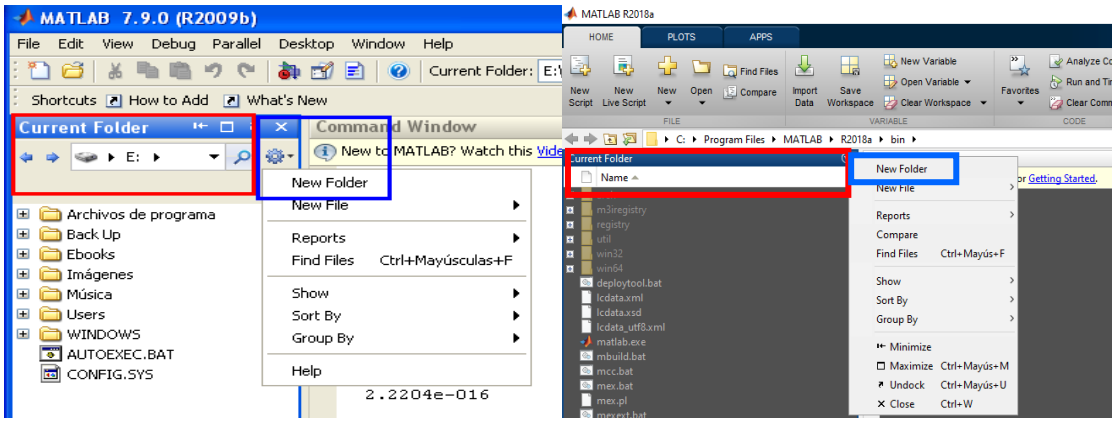

Figura 10. Ilustración para cambio de directorio para dos versiones de Matlab

2.3.- Los distintos tipos de ficheros en Matlab

El programa nos ofrece, entre otros, la posibilidad de trabajar con diferentes tipos de ficheros, como ser **M-file** o de **Figure**. Más adelante iremos desarrollando cada uno de estos ficheros y sus funciones. A modo introductorio decir que:

- En los ficheros **Figure** se guardarán las figuras que el programa nos permite realizar. Serán ficheros con extensión .fig.
- Los ficheros **M-Fife** tienen gran utilidad en el programa. Son ficheros que contienen conjuntos de comandos o definición de funciones y al teclear su nombre en la línea de comandos y pulsar ENTER, se ejecutan uno a uno todos los comandos contenidos en él. Siempre y cuando dicho fichero se encuentre en el directorio actual o esté incluido en el Path de Matlab.
- Existen otros ficheros de gran importancia, los de extensión **.mat** donde se guardarán las variables deseadas que han sido creadas durante el trabajo.

2.4.- Tipos de archivos

Gran parte de la funcionalidad de MATLAB se basa en su biblioteca de funciones (*toolbox* de MatLab). Una función en MATLAB es equivalente a una función matemática; es una tarea encapsulada que puede depender de una o varias variables. MATLAB tiene una extensísima biblioteca de funciones, la mayoría de ellas son archivos con la extensión *.m* que lee el programa.

Si en la consola introducimos:

>>sin(x)

¿Cómo sabe MATLAB dónde se encuentra la función seno? La respuesta es que MATLAB ya sabe en qué directorios del sistema puede encontrar archivos **.m** desde su instalación. ¿Significa esto que si creamos nuestras funciones debemos guardarlas en estos directorios? Ni mucho menos. MATLAB cuenta con un directorio especial en el que también busca funciones; es el llamado directorio de trabajo (directorio actual). Cada vez que se invoque una función en MATLAB buscará en los directorios habituales y en el directorio de trabajo.

MATLAB puede leer y escribir varios tipos de archivos, existen cinco tipos de archivos para almacenar datos o programas que son frecuentemente empleados.

1. **m-files** son archivos de texto estándar en código *ascii* con la extensión punto m, pueden ser *script files* o *function files*.
2. **mat-files** son archivos con extensión ***.mat***; son creados por MATLAB cuando salva información. Con el comando *save* los datos son guardados en este formato especial (binarios, no es posible editarlos desde un block de notas) que MATLAB puede leer estos archivos (se abran con el comando *load*)
3. **fig-files** son archivos de figuras con la extensión ***.fig***; pueden ser creados usando el comando *saveas* en el *Command Windows*, un archivo ***.fig*** contiene toda la información necesaria para recrear de nuevo una figura, estos archivos se pueden abrir con el comando *open* seguido del nombre del archivo y su extensión, por ejemplo *open ejemplo.fig* ,donde *ejemplo* es el nombre de la figura que queremos abrir
4. **p-files** son archivos compilados que tiene una extensión ***.p*** y que pueden ser ejecutados en MATLAB directamente. Son creados con el comando ***pcode*** y se emplean cuando se desarrolla un programa que otras personas puedan utilizar, pero no se quiere que se tenga acceso al código fuente (es decir al archivo ***.m***)
5. **mex-files** archivos de extensión ***.mex***; se utilizan cuando se llama código hecho en lenguaje C desde MatLab. Esto aumenta la velocidad de ejecución.

2.5.- Comandos básicos

Para el manejo de cualquier sesión MATLAB es muy común el uso de unos comandos específicos que nos ayudan a gestionar dicha sesión

1. **who** comando que hace una lista de las variables que existen en el Workspace
2. **whos** comando similar a **who** pero hace una lista de las variables de una forma larga, con información sobre su tamaño los bytes que ocupan su clase etc.
3. **home** comando que sirve para llevar el cursor a la primera línea de la ventana, pero manteniendo todos los comandos anteriores
4. **clc** (clear console) lleva el cursor a la primera línea y además limpia la pantalla, es decir se borran todos los comandos anteriores. **clc** sólo borra lo mostrado en el command windows pero las variables que han sido creadas siguen existiendo en la memoria y por lo tanto se pueden ver en el workspace y seguir trabajando con ellas
5. **clear** comando que elimina todas las variables que existen en workspace

6. **help** además de las ayudas que nos muestra MATLAB a través de su apartado HELP, también es posible acceder a las ayudas de las funciones mediante comandos entrados en command window, a través del comando **help** seguido del nombre de una función. Muestra un texto de ayuda de dicha función en la misma ventana
7. **helpwin** comando igual al **help** pero muestra un texto de ayuda en una ventana independiente, por fuera del command Windows.
8. format comando que permite editar el formato numérico con el que MATLAB muestra los resultados en pantalla, se usa tecleando la palabra format seguido el formato que deseemos, por ejemplo, ***format short***, ***format long***, etc.
9. **quit, exit** comandos que permiten salir de MatLab, funcionan exactamente igual.
10. **^C (Ctrl + C)** comando que interrumpe el programa localmente.
11. **what** comando que lista los M-files (archivos creados en MatLab) existentes en el directorio de trabajo.
12. **close** comando que cierra las ventanas de gráficas.
13. **clf** comando que limpia las ventanas de gráficos.

3.- Uso de los comandos básicos

El programa trabaja con diferentes tipos de números y expresiones numéricas: números enteros, racionales, reales y complejos

3.1.- Operaciones básicas

Comenzaremos realizando con el programa las operaciones más básicas entre números. Trabajamos en la ventana de Command window y para ello se utilizan las convenciones matemáticas estándar.

Por ejemplo, para sumar (Figura 11):

```
>> 2+2
ans 4
```

En general las operaciones básicas a realizar se hacen según la siguiente tabla y con la jerarquía habitual entre ellas:

a+b	Suma
a-b	Resta
a*b	Producto
a¨/b	División
a^b	potencia

Por defecto MATLAB trabaja con unos dígitos de aproximación para números decimales. Esto puede cambiarse.

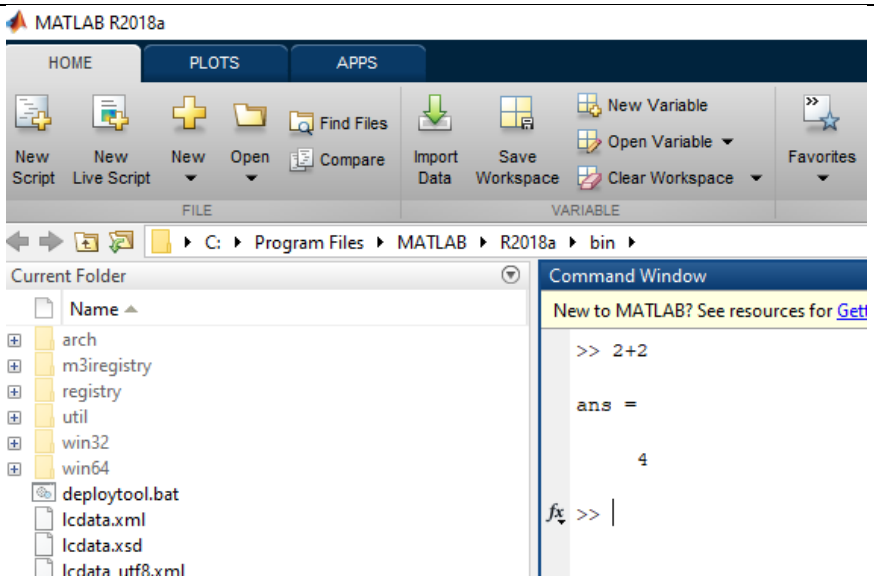

Figura 11 Operación básica suma

3.2.- Estableciendo la precisión de los cálculos

MATLAB trabaja siempre en doble precisión, es decir, guardando cada dato en 8 bytes, con 15 cifras decimales exactas. El formato con el que los datos aparecen en pantalla es variable. Existen varios modos de trabajo, se indican a continuación los más significativos:

format short	punto fijo con 4 decimales
format long	punto fijo con 14 decimales
format short e	4 decimales y forma exponencial.
format long e	15 decimales y forma exponencial.
format rat	formato racional.

El modo de trabajo por defecto en MATLAB es format short.

Ejemplo

```
>> 2^100
ans =
  1.2677e+30
>> 2+pi
ans =
  5.1416e+00
>> format long e
>> 2^100
ans =
   1.267650600228229e+30
>> 2+pi
ans =
   5.141592653589793e+00
>> format rat
>> 2+pi
ans =
   581/113
```

3.3.- Números complejos

MATLAB integra perfectamente el trabajo con números complejos. La letra minúscula *i* ó *j* representa el número imaginario$\sqrt{-1}$ (la unidad imaginaria). De esta manera los números complejos se representan mediante expresiones del tipo $a + ib$ ó $a + jb$.

Existen funciones específicas que actúan sobre los números complejos:

real(z): obtiene la parte real de z.

imag(z): obtiene la parte imaginaria de z.

conj(z): obtiene el complejo conjugado de z.

abs(z): obtiene el módulo de z.

angle(z): obtiene el argumento de z.

3.4.- Variables

En MATLAB es posible definir diferentes tipos de variables, y utilizando el comando *whos* es posible visualizar en la ventana de trabajo qué tipo de variables están almacenadas en la memoria de trabajo. Las variables en MATLAB tienen forma matricial: fila y columna.

3.4.1.- Variables numéricas

Si trabajamos con una variable unidimensional se crea con la forma de una fila y una columna. Para hacerlo sólo debemos teclear el nombre elegido e igualarlo a su valor:

```
>> a=8
```

y por defecto son de doble precisión. Pueden ser reales o complejas. Ejemplos:

- >> a = 25.4591
- >> b = 3.5+45*i, donde i = $\sqrt{-1}$ es la unidad imaginaria.

3.4.1.1 Alterar el valor de una variable:

MATLAB guarda el valor de la variable ejecutada en último lugar, es decir si volvemos a ejecutar un valor para "**a**" éste será el que mantiene. Podemos alterar el valor de una variable desde el workspace. Para ello cliqueamos en ella en la ventana del workspace y cambiaremos su valor desde el editor.

3.4.1.2 Guardar variables y recuperarlas

Normalmente es de gran interés guardar los valores de las variables con las que se ha trabajado en una sesión. Bien porque debe interrumpirse la misma y quiere recuperarse más adelante, o bien para utilizarse en nuevos trabajos relacionados con el que se ha realizado.

La forma más básica de guardar las variables es a través del botón de guardar situado en la ventana del workspace o en el botón Save Workspace (de las versiones más nuevas de Matlab). En cualquier versión se puede desplegar un menú al cliquear con el botón derecho del ratón sobre la variable situada en el workspace (Figura 12)

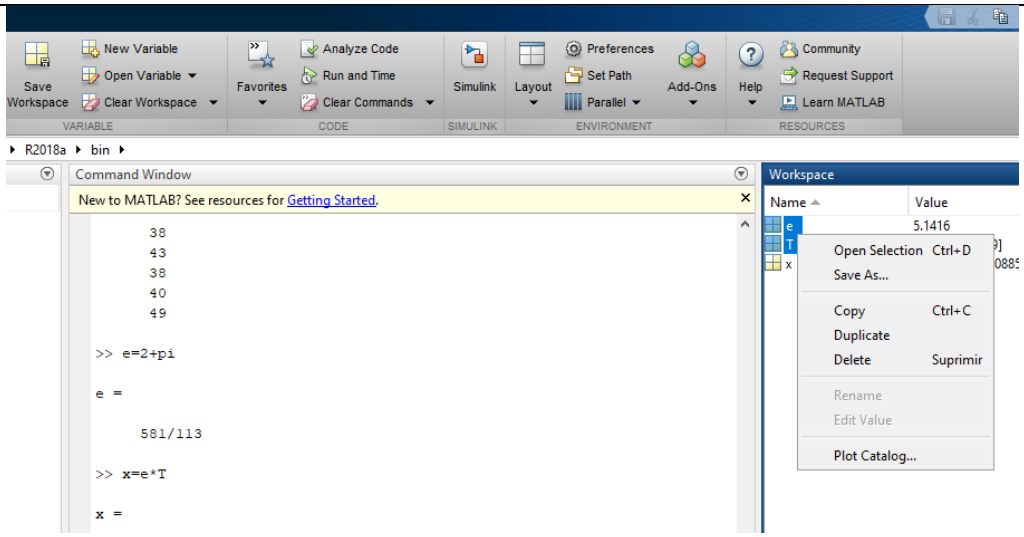

Figura 12 Salvar variables

Realizar cualquiera de estos procesos nos permite crear un fichero de extensión **.mat** con el nombre deseado cuyo contenido serán las variables que se quieren guardar . Al hacerlo, si guardamos en el directorio actual aparecerá un fichero en la lista (Current Folder) que se encuentra disponible para su utilización. Si en una nueva sesión queremos recuperar dichas variables sólo tenemos que cargar el fichero **.mat**, bien cliqueando dos veces sobre su nombre en la ventana del directorio actual, o arrastrándolo desde el Current Folder a la ventana Command Window o bien a través del menú File (Home), submenú (botón) Open.

3.4.1.3 Guardar sesiones de trabajo

Existen varios caminos para guardar lo realizado en una sesión de trabajo o en parte de ella, una forma es copiar la parte que nos interese del Command Window y abrir un fichero **.m**. (Figura 13). En ella podemos pegar lo seleccionado y modificar cuantas cosas queramos. Este fichero se guarda y se puede acceder a él cuando se desee. Se debe tener en cuenta que según lo realizado se trata de un fichero de texto, no es ejecutable.

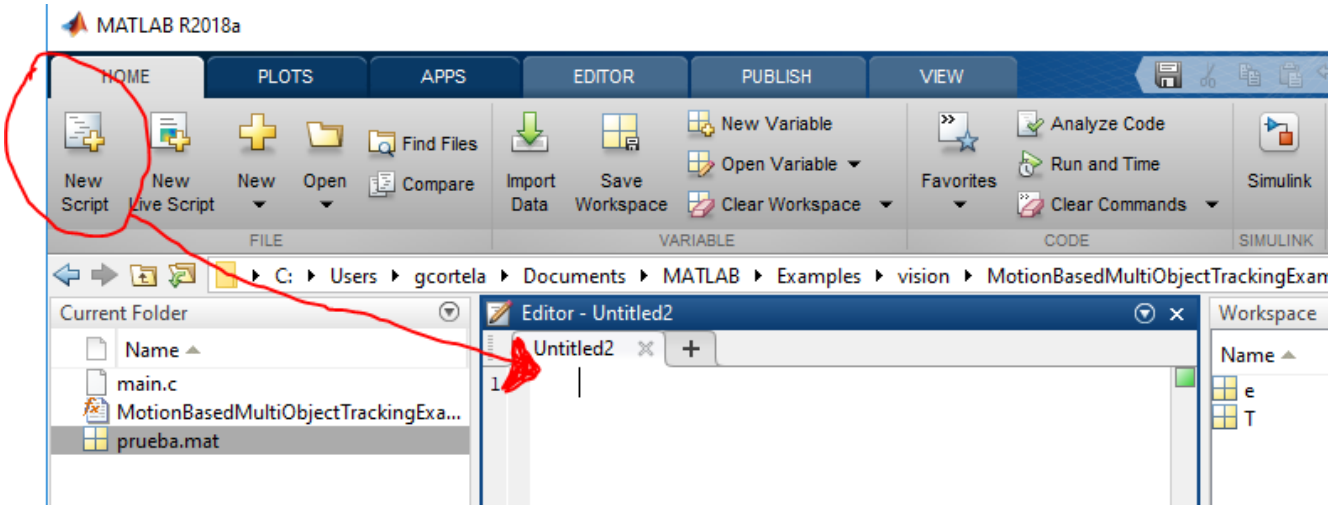

Figura 13 Abrir un script para salvar un fichero **.m**

3.4.2.- Variables string o carácter

Una cadena de caracteres determina una variable carácter. Su sintaxis es: a=’cadena de caracteres’

Como vemos es necesario incluir los caracteres entre comillas. Se destacan algunos comandos importantes que manejan este tipo de variables y que se irán usando en temas sucesivos:

str2mat(x1,x2,...):	Forma una matriz cuyas filas son las cadenas de caracteres x1, x2,..., respectivamente.
str2num(x1,x2,...):	Convierte la cadena de caracteres en su valor numérico exacto utilizado por Matlab.
num2str(número):	Convierte el número exacto en su cadena de caracteres equivalente con la precisión fijada
eval(expresión):	Ejecuta la expresión.
disp(‘cadena’):	Muestra la cadena tal y como se ha escrito y continua el proceso.
input(`cadena’):	Muestra la cadena en pantalla y espera que se presione una tecla para continuar

Ejemplos:

```
>> num2str(pi)
ans =
3.142
```

Se ha convertido el número exacto pi en cadena de caracteres.

```
>> str2num(‘15/14’)
ans =
1.0714
```

Se ha convertido una cadena a su valor exacto con la precisión por defecto.

```
>> str2mat(‘Canal 4’,’TNU’,’Canal 10’,’Canal 12’,)
ans =
Canal 4
TNU
Canal 10
Canal 12
```

Se ha formado la matriz de texto cuyas filas son las cadenas introducidas como los argumentos de str2mat.

```
>> disp(‘hola’)
ans =
hola
```

Se ha obtenido en pantalla la palabra escrita como argumento del comando.

3.4.3.- Variables simbólicas

Se definen luego del comando *syms*, por ejemplo: “*syms x y*” define las variables simbólicas *x* e *y*. Este tipo de variable permite realizar operaciones de forma simbólica, por ejemplo, resolver sistema de ecuaciones, calcular derivadas y/o integrales de funciones, etc. (Este tipo de variables no serán objeto de estudio durante el curso).

Notar que

Todos estos tipos de variable pueden definirse unitariamente o en forma de vectores o matrices, que contengan varias variables del mismo tipo. Existen otros formatos de variable que permiten mezclar variables de distinto tipo en una sola (por ej.: celdas).
Por más información sobre este tema consultar el **help** de MatLab: ***Data Types in MatLab.***

4.- Arrays

El array es una estructura fundamental que MATLAB utiliza para almacenar y manipular datos. Es una lista de números dispuestos en filas y/o columnas. En un array unidimensional los números están agrupados en filas o columnas, a diferencia de los arrays bidimensionales, donde los elementos se distribuyen en filas y columnas. En los arrays se puede almacenar información de manera similar a una tabla.

En las disciplinas científicas, a los arrays unidimensionales se les denomina comúnmente vectores, mientras que a los bidimensionales se les denomina matrices. Además de los arrays numéricos, en MATLAB los arrays pueden estar compuestos por caracteres; en este caso se denominan cadenas de caracteres o strings.

Notar que

El programa MATLAB se maneja (en su mayor parte) escribiendo sentencias dentro de una ventana de comandos. Los comandos o líneas de comando se escriben una a una (ya se verá la forma de escribir conjuntos de comandos o programas más adelante) pulsando la tecla de ***Enter*** al final. Por ejemplo, si se escribe sqrt(16) el programa realiza la operación indicada y responde en la pantalla con el resultado. Lo que se aprecia en la ventana de comandos será algo como:

```
>> sqrt(16)
ans =
4.0000
```

Las operaciones se indican de forma muy intuitiva, por ejemplo, para obtener el resultado de

$$(13\times 5)/(3+4\times 11)$$

basta con escribir 13*5/(3 +4*11). En la ventana de ordenes se obtendrá lo siguiente:

```
>> 13*5/(3 + 4*11)
ans =
1.3830
```

Observe que es necesario utilizar paréntesis para que la operación se lleve a cabo correctamente.

5.- Ingreso de Datos

Ya que el objeto básico de MATLAB son las matrices, es necesario conocer la forma de introducir datos que corresponden a los elementos de éstas. Las matrices pueden ser reales, complejas, de caracteres, etc. En general las matrices de 1x1 se interpretan como escalares y las de un sólo renglón (fila) o columna como vectores. Las matrices se pueden definir de alguna de las siguientes formas (no son las únicas):

- Por medio de una lista explícita de elementos
- De archivos externos (“cargándolas” con el comando “*load*” por ejemplo)
- Generadas a través de un proceso o rutina

Ejemplos de escritura directa:

```
>>A = [ 1 2 3; 4 5 6; 7 8 9 ]      % en este caso
                                   % el ; cambia de renglón

A = [
1 2 3
4 5 6
7 8 9 ]                            % en este caso el cambio de reglón
                                   % lo hacemos nosotros directamente
```

las dos formas anteriores son equivalentes

Ejemplo de lectura de un archivo externo con datos en forma de tabla ascii

```
>> load A.txt          % en este caso el archivo "A.txt" se encuentra en el directorio de
                       % trabajo o en la ruta de acceso o path
```

6.- Operadores y Funciones básicas

MATLAB puede operar las matrices por medio de operadores y por medio de funciones. Tiene predefinidas un conjunto de funciones matemáticas básicas. En el **help** hay información detallada de sintaxis para las mismas: ***>>help MatLab/elfun.***

+	adición o suma
–	sustracción o resta
*	multiplicación
'	transpuesta
^	potenciación
\	división-izquierda
/	división-derecha
.*	producto elemento a elemento
./ y **.**	división elemento a elemento
.^	elevar a una potencia elemento a elemento

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias. Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño. Si los operadores no se usan de modo correcto se obtiene un mensaje de error.

Un caso muy particular en MATLAB es el operador división derecha e izquierda ya que puede ser usado para resolver las siguientes ecuaciones:

x = A\b	es la solución de **A * x = b** (donde ***b*** es un vector columna)
x = b/A	es la solución de **x * A = b** (donde ***b*** es un vector fila)

Ejemplo:

```
[ 1 2 3 4 ].* [ 1 2 3 4 ] = [ 1 4 9 16 ]
```

que sería lo mismo que:

```
[ 1 2 3 4 ].^2 = [ 1 4 9 16 ]
```

y que no es lo mismo que:

```
[ 1 2 3 4 ]* [ 1 2 3 4 ]
```

porque obtendríamos un error:

```
Error using  *
Inner matrix dimensions must agree.
```

pero sí podríamos hacer:

```
[ 1 2 3 4 ]'* [ 1 2 3 4 ] =
1 2 3 4
 2 4 6 8
 3 6 9 12
 4 8 12 16
```

ya que al tomar la transpuesta de la primera matriz (o de la segunda) la operación es concordante. Nótese que tomando la transpuesta de la segunda matriz se obtiene, como es de esperar:

```
[ 1 2 3 4 ]* [ 1 2 3 4 ]' = 30
```

A continuación, veremos algunos ejemplos de funciones:

a) **Trigonométricas**. MATLAB tiene la posibilidad de trabajar con las funciones trigonométricas en radianes o grados. Las funciones trigonométricas terminadas en 'd' indican su versión en grados. Por ejemplo:

```
>> a = sin(pi/3) = 0.8660
>> b = sind(60) = 0.8660
>> c = asin(1) = 1.5708
>> d = asind(1) = 90
```

b) **Exponenciales**:

```
>> a = exp(5) = 148.4132
>> b = log(3) = 1.0986          Logaritmo natural (en base e)
>> c = log10(100) = 2           Logaritmo en base 10
```

c) **Números complejos:** consideremos el siguiente número complejo**:** $y = 5 + 2.3i$ [3]:

```
>> a = real(y) = 5
>> b = imag(y) = 2.3
>> c = abs(y) =  5.5036
>> d = conj(y)  = 5.0000 - 2.3000i
```

d) **Aproximaciones y redondeo**: x = [45.23 -23.79 0.3];

```
>>round(x) = [45 -24 0];      Redondea hacia el entero más próximo
>> fix(x) = [ 45 -23 0];      Redondea hacia cero
>> floor(x) = [45 -24 0];     Redondea hacia menos infinito
>> ceil(x) = [46  -23  1];    Redondea hacia más infinito
```

7.- Notación con el uso de dos puntos, o colon (:)

En MATLAB el símbolo **:** (dos puntos) juega un papel muy importante. Se usa para definir rangos de una variable o matriz o para indicar la totalidad de una columna o renglón, o el contenido total de una matriz.

Ejemplos:

```
A ( 1:3 , 4:6 ) % proporciona la submatriz definida por
                % los renglones 1 a 3 y por las columnas 4 a 6
x = [ 0.0 : 0.1 :1.0]    % proporciona un vector renglón que
                % va desde 0.0 hasta 1.0 en incrementos de 0.1
A ( 2:5 , 4 )   % nos da la los elementos 2 al 5, de la columna 4
A ( : , 3 )     % nos da la todos los elementos de la columna 3

A(:, [2 4 5]) = B(: , 1:3)      % cambia las columnas 2, 4 y 5
                                % de A por las tres primeras
                                % columnas de B
```

[3] También se puede escribir el complejo usando j como unidad imaginaria: **y=5+2.3*j**

8.- Las matrices en MatLab

Como antes se comentó, una de las características de MATLAB es que esta especialmente diseñado para trabajar con variables vectoriales y matriciales.

8.1.- Construcción de matrices.

Veamos ahora cómo construir vectores y matrices en **MatLab**. Esto es fundamental pues los datos deben ser presentados en forma de matrices para que el programa pueda procesarlos.

De esta forma podemos **seleccionar un elemento dado de una matriz** por la fila n y la columna m a la cual pertenece:

```
>> a = A(n,m);
```

De la misma manera, se puede seleccionar el n-ésimo elemento de un vector (por ejemplo, **d**) como **d(n)**. Al seleccionar elementos, el parámetro *end* se puede usar para referirse al último elemento de un vector. Así **d(end)** selecciona el último elemento de **d**. Cabe notar que entre los paréntesis incluso pueden referenciarse operaciones u otras variables.

size	Dimensiones de las matrices: **size(A)** devuelve un vector con dos componentes enteros, el número de filas y el número de columnas.
[fila,columna] =size(A)	Devuelve los dos enteros en variables separadas.
fila=size(A,1)	Devuelve la dimensión de las filas
columna=size(A,2)	Devuelve la dimensión de las columnas.

En el siguiente ejemplo definimos una matriz **A** de 2 filas y 3 columnas (es decir que tiene dimensiones 2×3), escribiendo los elementos que constituyen sus filas.

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

El comando *size* proporciona las **dimensiones de la matriz** (el primer número corresponde a la cantidad de filas y el segundo a la cantidad de columnas):

```
>> g = size(A)
g =
2    3
```

Si por ejemplo quisiéramos identificar el elemento ubicado en la fila 2 y la columna 1 de la matriz A, ejecutamos:

```
>> e21 = A(2,1)
e21 =
6
```

Podemos hacer esta asignación

```
>> a=[2 3 0 1];
```

sin haberle indicado previamente al programa que **a** no es una variable escalar (es decir, una variable en la que almacenamos un solo número) sino una variable vectorial. De hecho, en MATLAB no hay propiamente variables

numéricas escalares ni vectoriales, sino matriciales (arrays): si se mira el Workspace en cualquier sesión de trabajo se verá que los números se van almacenando como matrices 1×1. Análogamente, nuestra variable **a** es para MATLAB una matriz 1 × 4.

Las matrices se introducen entre corchetes (**[ ]**), separando las filas por **;** y los elementos de cada fila por comas o simplemente espacios.

```
>> A=[0 -1 3 2; 2 1 7 2; 3 0 6 3; 5 0 10 6]
A =
     0  -1   3   2
     2   1   7   2
     3   0   6   3
     5   0  10   6
```

Como no hemos puesto **;** al final de la introducción de datos, MATLAB nos contesta con el valor de la variable. Tanto en la ventana de comandos como en la de variables, ya aparece colocada en forma matricial.

Las variables **a** y **A** no se interfieren (coexisten en el Workspace) porque MATLAB distingue mayúsculas de minúsculas. Las variables pueden estar formadas por varios caracteres (como ya hemos visto con los ejemplos de **ans** y **pi**), pero el primero de ellos siempre ha de ser una letra.

Vamos a crear dos variables matriciales más (fijaos en que todas van apareciendo en la ventana del Workspace):

```
>> D=[2 -1 3 0 ; 0 0 1 5]
D =
        2  -1  3  0
        0   0  1  5

>> E=rand(4,4)
E =
     0.9501   0.8913   0.8214   0.9218
     0.2311   0.7621   0.4447    0.7382
     0.6068   0.4565   0.6154   0.1763
     0.4860   0.0185   0.7919   0.4057
```

(El comando rand crea una matriz del tamaño especificado, en este caso 4 × 4, formada por números aleatorios con distribución uniforme entre 0 y 1.)

Las operaciones de suma de matrices y producto de un escalar por una matriz se realizan directamente, sin necesidad de ir componente a componente:

```
>> A+E
ans =
        0.9501   -0.1087    3.8214   2.9218
        2.2311    1.7621    7.4447   2.7382
        3.6068    0.4565    6.6154   3.1763
        5.4860    0.0185   10.7919   6.4057

>> 3.5*E
ans =
        3.3255  3.1195  2.8749  3.2263
        0.8090  2.6673  1.5565  2.5837
        2.1239  1.5976  2.1540  0.6169
        1.7009  0.0648  2.7718  1.4200
```

Por supuesto, si intentamos sumar dos matrices de tamaños distintos obtendremos un mensaje de error

```
>> A+D
Error using  +
Inner matrix dimensions must agree.
```

Igual de fácil resulta multiplicar matrices

```
>> D*E
ans =
    3.4896   2.3899   3.0444   1.6342
    3.0368   0.5490   4.5751   2.2048
```

D*E es el producto ordinario de las matrices D y E. Para que tenga sentido la operación, el número de columnas del primer factor tiene que coincidir con el número de filas del segundo

```
>> E*D
Error using  *
Inner matrix dimensions must agree.
```

8.2.- Formas de definir vectores

En MatLab, un vector se crea asignando sus elementos a una variable. Esto se puede hacer de distintas formas, dependiendo del origen de la información utilizada para componer los elementos del vector. Cuando un vector contiene números que son conocidos (por ejemplo, coordenadas de un punto), el valor de cada elemento se introduce directamente. Cada uno de estos elementos puede ser también una expresión matemática con variables predefinidas, números y funciones. A menudo, los elementos de un vector fila son series de números con distancia o espaciado constante. En estos casos el vector puede ser creado a partir de comandos MatLab. Un vector se puede crear también a partir del resultado de una operación matemática.

8.2.1.- Creación de un vector a partir de una lista de números conocidos:

Para crear el vector sólo es necesario teclear sus elementos (números) dentro de un par de corchetes [ ].

nombre_variable = [elementos del vector]

Vector fila: para crear un vector fila deben teclearse los elementos con un espacio o una coma entre cada elemento, dentro de los corchetes.

Vector columna: para crear un vector columna deben teclearse los elementos con un punto y coma entre cada elemento, o pulsando la tecla ***Enter*** después de cada elemento dentro de los dos corchetes. También se puede definir como vector fila y luego trasponer (con una comilla simple al final).

8.2.2.- Creación de un vector con distancia constante a partir de la especificación del primer término, de la distancia y del último término:

En un vector con distancia o espaciado constante, la diferencia entre los elementos es siempre la misma. Por ejemplo, en el vector: *v* = 2 4 6 8 10, la distancia entre los elementos es 2. Un vector donde el primer término es m, el espacio o distancia es ***q***, y el último término es ***n***, se puede crear a partir de la expresión

nombre_variable = [m:q:n]

o

nombre_variable = m:q:n

Si los números ***m***, ***q*** y ***n*** son tales que el valor de ***n*** no se puede obtener añadiendo las diferencias ***q*** a ***m***, entonces (para un ***n*** positivo) el último elemento del vector será el último número que no excedan.

8.2.3.- Creación de un vector con distancia constante a partir de la especificación del primer y último término y el número de términos

Un vector en el cual el primer elemento es x_i , el último elemento es x_f , y el número de elementos es ***n***, puede ser creado utilizando el comando *linspace*. MATLAB determina la distancia correcta entre los elementos

nombre_variable =linspace (xi,xf,n)

El número de elementos N del vector, se relaciona con n_i, y n_f mediante la expresión:

$$N = \frac{nf - ni}{dn} + 1$$

Como ejemplo definamos un vector **v** que contenga los enteros 3, 4 y 5 en ese orden:

```
>>v=[3  4  5]
v=
        3   4   5
```

Otra forma de definir el vector anterior sería de la siguiente forma:

```
>> v=[3:5];
```

En general se puede definir un vector cuyos elementos se obtienen a partir del anterior más un cierto incremento dn con la sintaxis, por ejemplo si queremos crear un vector v que comienza en n_i, se incrementa con dn y termina en n_f :

>> v=[n_i:dn:n_f].[4]

En el caso que $dn = 1$ alcanza con escribir

>> v=[n_i: n_f]

Si queremos crear un vector $\boldsymbol{u}$, que comience con n_i, termine en n_f, y tenga N elementos se debe usar el comando ***linspace*** y usar la siguiente sintaxis: >> $linspace(n_i, n_f, N)$, o calcular el dn correspondiente.

8.3.- Creación de arrays bidimensionales, matrices.

Un array bidimensional, también llamado matriz, distribuye los números en columnas y filas (almacena información como si fuera una tabla). En una matriz cuadrada, el número de filas y columnas coincide. Por ejemplo, la matriz:

1 2 3
4 5 6
7 8 9

[4] Previamente se deben haber definido las variables n_i y n_f, es decir haberles asignado un valor numérico.

es cuadrada (3×3), ya que está formada por tres filas y tres columnas. En general, el número de filas y de columnas puede ser diferente. Por ejemplo, la matriz:

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

tiene tres filas y cinco de columnas. En general la matriz *m*×*n* tiene *m* filas y *n* columnas, además el producto de *m* por *n* nos da el tamaño de la matriz (número elementos).

Para crear una matriz sólo hay que asignar los valores correspondientes a una variable. Para hacer esto es necesario seguidamente teclear los elementos fila por fila, entre corchetes [ ]. Primero se escribe el corchete izquierdo **[**, seguidamente se teclea la primera columna separando los elementos elementos por espacio o comas. Antes de teclear la siguiente fila, hay que poner un punto y coma o pulsar ***Intro***. Finalmente se pone el corchete derecho **]** al final de la última fila.

nombre_variable = [elementos de la 1ª fila; elementos de la 2ª fila;
elementos de la 3ª fila; …….; elementos de la última fila]

Los elementos de una matriz pueden ser números o expresiones matemáticas que incluyan números, variables predefinidas y funciones. Todas las filas deben tener idéntico número de elementos.

```
>> A = [1:2:11; 0:5:25; linspace(10,60,6); 67 2 43 68 4 13]
A =
```

$$\begin{matrix} 1 & 3 & 5 & 7 & 9 & 11 \\ 0 & 5 & 10 & 15 & 20 & 25 \\ 10 & 20 & 30 & 40 & 50 & 60 \\ 67 & 2 & 43 & 68 & 4 & 13 \end{matrix}$$

En este ejemplo, las primeras dos filas se introducen como vectores utilizando la notación de espaciado constante. La tercera fila se introduce utilizando el comando *linspace*, mientas que en la última fila se introducen los elementos individualmente.

Existen comandos para crear matrices que contendrán elementos con valores especiales: *zeros (m, n)*, *ones* (m, n) y *eye (n)*. Los comandos *zeros(m,n)* y *ones(m,n)* crean matrices de *m* filas y *n* columnas en las que todos los elementos son ceros y unos respectivamente. El comando *eye(n)* crea una matriz cuadrada (*n* filas y *n* columnas) en la cual los elementos de la diagonal son unos, siendo ceros el resto de los elementos, llamada matriz identidad.

Un vector también puede definirse a partir de los elementos de una fila o columna de una matriz dada. En el siguiente ejemplo los elementos del vector **v** corresponden a la segunda fila de la matriz **A**, y los elementos del vector **w** corresponden a la primera columna de la misma matriz:

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

>> C = [A(:,2), A(:,1)]
C =
    4    3
    7    6
    2    1
```

Ejecutando el comando *length* podemos obtener la **longitud de un vector dado:**

```
>> L = length(w)
        L  =  3
```

En el caso de una matriz, el comando *length* proporciona la cantidad de columnas, existe el comando *numel* da la cantidad de elemento del array.

8.4.- Operaciones con arrays.

Los elementos dentro de un array (ya sea un vector o una matriz) pueden ser manipulados individualmente o en grupo. Esto es especialmente útil cuando se necesita redefinir sólo algunos de los elementos para ser utilizados en cálculos específicos, o cuando un subgrupo de elementos se utiliza para definir una nueva variable.

8.4.1.- Vectores

La dirección de un elemento en un vector es su posición en la fila (o columna). Si tenemos un vector llamado ***u***, ***u(k)*** refiere al elemento de ***u*** en la posición ***k***. La primera posición es l. Por ejemplo, si el vector ve tiene nueve elementos:

u =[35 46 78 23 5 14 81 3 55]

u(4)=23, u(7)=81, y u(1)=35

Un elemento de un vector ***u(k)*** se puede utilizar como una variable. Por ejemplo, es posible cambiar el valor de un solo elemento de un vector reasignando un nuevo valor a la dirección específica del elemento en cuestión. Esto se hace tecleando: ***u(k) = valor***. Un elemento de un vector puede ser utilizado también corno variable en expresiones matemáticas.

```
>>u = [35 46 78 23 5 14 81 3 55]
u=
35  46  78  23  5  14  81  3  55
>>u(2)+ u(8)
ans=
49
>>u(5)^u(8)+ sqrt(u(7)
ans=
134
>>u(2)= -19
u=
35  -19  78  23  5  14  81  3  55
```

8.4.2.- Matrices

La dirección de un elemento en una matriz es su posición definida a partir del número de fila y de columna dentro de la propia matriz. Si tenemos una matriz ***ma***, el elemento ***ma(k, p)*** se refiere al que ocupa la fila ***k*** y la columna ***p***.

Por ejemplo, sea $ma = \begin{bmatrix} 3 & 11 & 6 & 5 \\ 4 & 7 & 10 & 2 \\ 13 & 9 & 0 & 8 \end{bmatrix}$

Entonces *ma*(1,1)=3 y *ma*(2,3)=10. Al igual que sucede con los vectores, es posible cambiar el valor de un solo elemento de la matriz asignándole un nuevo valor. Así mismo, los elementos se pueden utilizar individualmente como variables en expresiones matemáticas y funciones. Ejemplo, si queremos cambiar el valor al elemento *ma*(3,1) de 13 a 20, operamos:

```
>>ma = [3 11 6 5; 4 7 10 2; 13 9 0 8]
ma=
   3  11  6  5
   4  7  10  2
   13  9  0  8
>>ma(3,1)=20
ma=
   3  11  6   5
   4   7  10  2
   20  9   0  8
```

Notar que

Todas las variables en MATLAB son arrays. Un escalar es simplemente un array de un solo elemento. Un vector es un array con una sola fila o una sola columna de elementos. Una matriz es un array con elementos distribuidos en filas y columnas.

Las variables (escalares, vectores o matrices) se definen a partir de la entrada, cuando a la variable se le asigna un valor. No hay necesidad de definir explícitamente el tamaño del array antes de que se produzca la asignación.

8.5.- Adición de nuevos elementos a variables ya creadas

Una variable que ya ha sido creada previamente puede alterar su estructura inicial mediante la inserción de nuevos elementos (un escalar es un vector de un solo elemento). Un vector (una matriz con una sola fila o columna) puede cambiar de tamaño para contener más elementos de los que ya tiene; e incluso puede convertirse en una matriz bidimensional. A las matrices también se les puede añadir filas y columnas para obtener un array de tamaño distinto de la inicial. Para añadir elementos a un array simplemente hay que asignar nuevos elementos a los ya existentes. También se pueden añadir variables previamente creadas.

Adición de elementos a un vector. Se pueden añadir elementos nuevos a un vector existente mediante una asignación de estos elementos. Por ejemplo, si un vector tiene 4 elementos, éste puede crecer de tamaño asignando nuevos números a las posiciones 5, 6, etc. Si un vector tiene, en general, ***n*** elementos y se le asigna un nuevo valor en la posición ***n+2*** o mayor, MATLAB asigna ceros a los elementos que hay entre el último elemento del vector original y el nuevo elemento añadido.

A una matriz existente se le pueden añadir nuevas filas y columnas asignándole valores a las nuevas filas o columnas dentro de la matriz. Nuevamente, esto se puede realizar asignando nuevos valores o añadiendo una variable predefinida. En cualquier caso, debe tenerse cuidado, ya que el tamaño de las columnas y filas añadidas debe coincidir con el de la matriz original

```
>>E= [l 2 3 4; 5 6 7 8]
E=
    1  2  3  4
    5  6  7  8
>>E(3,:) = [10:4:22]
E=
     1   2   3   4
     5   6   7   8
    10 14 18 22
>>K=eye(3)
K=
    1  0  0
    0  1  0
    0  0  1
>>G=[E K]
     1   2   3   4  1  0  0
     5   6   7   8  0  1  0
    10 14 18 22  0  0  1
```

Si una matriz tiene un tamaño de ***m***×***n***, y se le asigna un nuevo elemento a una dirección más allá del tamaño de la matriz, MATLAB incrementará el tamaño de la matriz para incluir el nuevo elemento. Así mismo se añadirían ceros en los huecos.

8.6.- Cadenas de caracteres y variables de tipo string

Una cadena de caracteres o string es simplemente un array de caracteres. Para crear una cadena solo es necesario teclear los caracteres que la forman entre comillas simples.

Las cadenas pueden incluir letras, dígitos, algunos símbolos y espacios. Sin embargo, y debido a la versión en inglés de MatLab. Son ejemplos de cadenas válidas: ‘ad ef’, ‘3%fr2’, ‘{edcba:21!’, ‘MatLab’, ‘annio’, ‘Jose’, ‘camion’, ‘año’, ‘José’ y ‘camión’.

Cuando comienza a teclearse una cadena, el color del texto en la pantalla cambiar a tonalidad púrpura, una vez que se teclea la primera comilla. Cuando se teclea la última comilla, al final de la cadena, el color de la cadena cambia.

Las cadenas de caracteres tienen diferentes usos en MATLAB. Normalmente se utilizan como salida para visualizar ciertos mensajes de texto, en comandos de formato para gráficos y como argumentos de algunas funciones.

Cuando las cadenas se utilizan en gráficos (etiquetas de ejes, título y notas de texto), los caracteres que forman la cadena pueden ser representados con una fuente, tamaño, posición, color, etc. específicos.

Ejemplos:

```
>>a= 'Mjfr6'
a=
    Mjfr6
>>A= 'Mi nombre es Juan'
A=
    Mi nombre es Juan
```

Cuando a una variable se le asigna una cadena, se almacena en memoria de la misma forma que sucede con los números. Cada carácter, incluido el espacio, se corresponde con un elemento en el array. Esto implica que una línea de caracteres es un vector fila en el cual el número de elementos se corresponde con el número de caracteres. A los elementos del vector se accede mediante sus posiciones. Por ejemplo, en el vector A que se definió antes, el cuarto elemento se corresponde con la letra ***n***, el decimocuarto elemento con la **J**, y así sucesivamente. Al igual que con los vectores que contienen números, es posible también cambiar los elementos de una cadena mediante asignación directa por la posición de los elementos. Por ejemplo, en el vector A que se definió antes, el nombre Juan puede ser reemplazado por José de la forma:

```
>>A(14:17)= 'José'
A=
     Mi nombre es José
```

Las cadenas también pueden almacenarse en una matriz. Al igual que sucede con los números, la forma de hacerlo es tecleando un punto y coma **;** (o pulsando la tecla *Enter*) al final de cada fila. En este caso cada fila debe teclearse como una cadena, es decir, debe estar encerrada entre comillas. Además, como sucede con las matrices de números, el número de elementos (longitud del string) en todas las filas debe ser el mismo. Este requisito puede causar algunos problemas cuando la intención del usuario es crear filas con palabras distintas. En este caso podrían utilizarse espacios para asegurarse que todas las filas tengan el mismo número de elementos.

MATLAB proporciona una función llamada *char* que crea un array con filas que tienen el mismo número de caracteres a partir de datos de entrada (en forma de filas) que no tienen por qué ser de la misma longitud. MATLAB hace que la longitud de todas las filas sea igual a la de mayor tamaño, a base de añadir espacios al final

de las líneas más cortas. La función *char* admite como parámetros de entrada las cadenas separadas por coma, según el formato:

nombre variable = char ('cadena 1', 'cadena 2', 'cadena 3', ... , 'cadena N')

```
>>Info = char('Nombre del Estudiante:','Luis','Curso:','Primero)
Info=
    Nombre del Estudiante:
    Luis
    Curso:
    Primero
```

Una variable se puede definir como un número o una cadena que tenga los mismos dígitos. Por ejemplo podemos definir a x como un número 536, e y definirla como una cadena que contiene los dígitos 536.

```
>>x= 536
x=
  536
>>y= '536'
y=
   536
>>
```

Utilización de los dos puntos(:)

Los dos puntos se utilizan para acceder a un rango de elementos dentro de un vector o una matriz.

En vectores ***u(:)*** se refiere a todos los elementos del vector ***u*** (ya sea un vector fila o columna).

u(m:n) se refiere a todos los elementos comprendidos entre las posiciones ***m*** y ***n*** del vector ***u***.

En matrices, **A(:,n)** se refiere a los elementos de la columna ***n*** de la matriz **A**. **A(n,:)** se refiere a los elementos de la fila **n** de la matriz **A**. **A(:,m:n)** se refiere a los elementos entre las columnas **m** y **n** de la matriz **A**. **A(m:n,:)** se refiere a los elementos entre las filas **m** y **n** de la matriz **A**. **A(m:n,p:q)** se refiere a los elementos de la fila **m** a la **n**, y a los de la columna **p** a la **q** de la matriz **A**. Mas adelante veremos su empleo en operaciones lógicas

Estas dos variables son completamente distintas, aunque parecen idénticas en la pantalla. La variable x puede ser usada en operaciones matemáticas, mientras que la variable y no.

8.7.- Operaciones matemáticas con array

Una vez creadas las variables en MatLab, se pueden utilizar para realizar operaciones matemáticas de distinta índole. Recordemos que un escalar se puede considerar como array de dimensión l×1 (array de una fila y una columna, es decir, con un solo elemento). Los arrays, no obstante, pueden ser de una dimensión (una fila o una columna), dos dimensiones (filas y columnas) o incluso tener dimensiones superiores. En estos casos las operaciones matemáticas son un tanto más complejas. MATLAB está especialmente diseñado para realizar operaciones avanzadas con arrays. Se presentan las operaciones matemáticas básicas con arrays que se pueden realizar.

8.7.1.- Suma y Resta.

En este caso ***las matrices deben tener las mismas dimensiones*.** Por ejemplo, definamos dos matrices $\mathbf{M_1}$ y $\mathbf{M_2}$, y luego efectuemos la suma entre ellas; comprobaremos en el siguiente ejemplo que cada elemento de la matriz suma es la suma de los elementos correspondientes de las matrices sumandas (ídem para la resta):

```
>> M1 = [1 2 3; 4 5 6];
>> M2 = [0 1 0;-1 2 1];
>> s = M1 + M2
        s =
          1    3    3
          3    7    7
```

8.7.2.- Producto entre matrices.

La operación de multiplicación ***** es ejecutada por MATLAB según las reglas propias del álgebra lineal. Esto significa que, si **A** y **B** son dos matrices, la operación **A*B** se ejecuta solamente ***si el número de columnas de la matriz A es igual al número de filas de la matriz B***. El resultado es una matriz que tiene el mismo número de filas que A y el mismo número de columnas que B.

Sean

$$A=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \text{ y } \quad B=\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

el producto $P = A * B$ viene dado por

$$P=\begin{pmatrix} a_{11}b_{11}+a_{12}b_{21}+a_{13}b_{31} & a_{11}b_{12}+a_{12}b_{22}+a_{13}b_{32} \\ a_{21}b_{11}+a_{22}b_{21}+a_{23}b_{31} & a_{21}b_{12}+a_{22}b_{22}+a_{23}b_{32} \end{pmatrix}$$

El resultado de multiplicar una matriz $A(m \times p)$ por otra $B(p \times n)$ es una matriz $C(m \times n)$, cuyos elementos se definen como:

$$C(i,j) = \sum_{k=1}^{p} A(i,k) * B(k,j) = A(i,1) * B(1,j) + A(i,2) * B(2,j) + \cdots + A(i,p) * B(p,j),$$

donde *i* puede tomar valores enteros entre 1 y *m*, y *j* puede tomar valores enteros entre 1 y *n*.

Veamos el siguiente ejemplo:

```
>> A = [3 4 5; 6 7 8];
>> B = [3 2; 1 1; 0 -3];
>> P = A*B
P =
   13   -5
   25   -5
```

El producto de la multiplicación de dos matrices cuadradas (deben ser del mismo tamaño) es también una matriz cuadrada del mismo tamaño. Sin embargo, la multiplicación de matrices no es conmutativa. Esto significa que si A y B son de dimensión ***n*×*n***, entonces A*B≠B*A (excepto para casos particulares como ser la matriz nula e identidad). Además, operaciones como la potencia sólo se pueden ejecutar con matrices cuadradas (ya que A*A sólo puede resolverse si el número de columnas de la primera matriz es igual al número de filas de la segunda matriz).

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

```
>> B = [3 2; 1 1; 0 -3];
>> P = B*A
P =
   21   26   31
    9   11   13
  -18  -21  -24
```

8.7.2.1 Producto de dos matrices elemento a elemento

En el caso que las matrices tengan iguales dimensiones. Se obtiene entonces una matriz donde cada elemento está definido por el producto de los elementos correspondientes en las matrices dadas. Por ejemplo:

```
>> A = [3 4 5; 6 7 8];
>> B = [1 -1 2; 0 2 4];
>> P = A.*B
        P =
          3   -4   10
          0   14   32
```

Notar que en este caso hay que anteponer el punto (.) al símbolo del producto (*) para que realice la operación elemento a elemento, si no se incluye (**P=A*B**) da un error.

Todo lo anterior referido al producto entre matrices es válido igualmente para el **cociente entre** matrices (siempre que el determinante de la matriz divisora sea no nulo), y también para el **producto y cociente entre vectores** (por ser el vector un caso particular de matrices).

8.7.3.- Potencia enésima de una matriz

(sólo posible para matrices cuadradas): `>> A^n`

8.7.4.- Potencia enésima de una matriz elemento a elemento

(en este caso la matriz NO tiene porque ser cuadrada): `>> A.^n`

8.7.5.- Productos entre vectores:

Sean los vectores $\boldsymbol{x}$ e $\boldsymbol{y}$:

```
>> x= [1 2 3];          >> y= [2 -1 4];
```

8.7.5.1 Producto escalar:

```
dot(x,y)  el resultado es un escalar:
>> prodescal = dot(x,y)
prodescal =
          12
```

8.7.5.2 Producto vectorial

```
cross(x,y)  el resultado es vector:
>> prodvect = cross(x,y)
prodvect =
```

11 2 -5

Ejemplos

E.1) Defina las siguientes matrices **a** = [3 8.2 -1]; **b** = [0.5 4 -3.2; 4 0.7 10]; **c** = [-3:0.1:25];

a) Defina una variable que indique las dimensiones de cada una de las matrices anteriores. Para las matrices **a** y **c**, defina una nueva variable que contenga el número de elementos.

b) Seleccione el segundo elemento del vector **a**, y el anteúltimo elemento del vector **c**.

c) Seleccione la segunda fila de la matriz **b** y su tercera columna.

d) Borre las variables de la memoria.

e) Limpie la pantalla.

E.2) Defina las siguientes matrices: **a** = [-3 4 5;2 0 0;2 0 -1]; **b** = 3*I(3), siendo I(3) la matriz identidad de 3x3; **c** = [1 3 -2+4*i;0-2*i 1 3;9 0 1];

Calcular:

a) **s** = a+b, **r** = a-c; **p** = a*c; **d** = a^2; **e** = a^(1/2) **g** = exp(a)

b) Producto y cociente elemento a elemento de las matrices **a** y **c**

c) Transpuesta de **a**

d) Matriz conjugada de **c**

e) Traza de **a**

f) Determinante de **a**

g) Inverso de **c**

Algunos caracteres especiales

[ ] Los corchetes se utilizan para componer vectores y matrices. [4 7 9] es un vector fila de tres elementos separados por blancos. [4; 7;9] es un vector de tres columnas. El punto y coma termina cada fila. Se permite el uso de vectores y matrices como elementos de un vector o matriz. Por ejemplo: [a b; c] siempre que el número de filas de a y b sean iguales y c tenga el mismo número de columnas que número de columnas de a más número de columnas de b.

a =[ ] crea una matriz vacía.

( ) Los paréntesis se utilizan para indicar precedencia en expresiones aritméticas. También se utilizan para encerrar los índices de vectores y matrices. Si el índice es menor que 1 o mayor que la dimensión, ocurre un error.

= Utilizado en sentencias de asignación.

' Traspuesta de la matriz. *x'* es la traspuesta conjugada, mientras que *x.'* es la traspuesta no conjugada de la matriz *x*.

. Punto decimal. Indicador de operaciones elemento a elemento.

... Tres puntos o más al final de una línea indican que ésta continuará en la línea de debajo.

, La coma se utiliza para separar índices de la matriz y argumentos de las funciones. También se utiliza para separar sentencias en líneas multisentencia. Ejemplo *sin(pi),cos(pi)*

; Dentro de corchetes finaliza una línea. Después de una expresión o sentencia suprime la impresión en pantalla de la misma, o la separa de otra.

% Denota un comentario. Indica el fin lógico de una línea. Cualquier otro texto posterior se ignora.

! Indica que el resto de la línea de entrada es un comando del sistema operativo.

: Se utiliza para crear vectores, como índice de matrices y para iteraciones.

Utilización de los dos puntos(:) en operaciones lógicas

El símbolo : (dos puntos) también puede usarse para definir una secuencia Booleana (condicional) y evitar la programación de bucles.
Definamos una matriz:

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

Si queremos obtener los elementos de la columna 3 de A que cumplen con una condición específica (ejemplo mayores o iguales a 5.0) podemos hacer:

```
>> i = A(: , 3) >= 5.0
 i =
     0
     1
     1
```

El vector "i" (identificador) contiene tantos elementos como hay en la columna 3 de A, con "ceros", en donde no se cumple la condición, y "unos" en donde sí se cumple. Se puede incluir el resultado de una operación lógica como "subíndices virtuales" de la matriz para obtener el resultado. Haciendo:

```
>> A(i,3)
ans =
 6
 9
```

A lo cual podríamos llamar con otro nombre y tener disponible el resultado, es decir

```
>> B = A(i,3)
B =
 6
 9
```

Ahora tendremos un vector B que contiene sólo los elementos de A que cumplieron con la condición.
Para hacer la comparación de la secuencia "normal" que habríamos tenido que seguir, el ejemplo anterior correspondería a lo siguiente:

```
>> i = A( : , 3 ) >= 5.0
 igual que en el 1er paso anterior
i =
0
 1
 1
```

Una vez obtenido el vector "*i*" podríamos usar la función "*find*" de MATLAB para obtener los subíndices de los elementos que no son cero:

```
>> subi = find(i)
subi =
  2
  3
```

Lo cual nos proporciona los subíndices de los elementos que cumplen con la condición, y los renombra como el vector "*subi*", si ahora hacemos:

```
>> A(subi,3)
          ans =
           6
           9
```

nos vuelve a dar el resultado esperado. Se puede hacer todo el ejemplo en una sola línea como sigue:

```
>> B = A(A( : , 3 ) >= 5.0,3)
B =
 6
 9
```

9.- Lectura y almacenamiento de datos.

9.1.- Almacenamiento de datos.

Usualmente será necesario interrumpir el trabajo con MATLAB y poder continuarlo más tarde, recuperando la sesión en el mismo punto en el que se dejó (con las mismas variables definidas, con los mismos resultados intermedios, etc.). Hay que tener en cuenta que al salir del programa todo el contenido de la memoria se borra automáticamente.

Para guardar el estado de una sesión de trabajo existe el comando *save*. Si se teclea: **>> save** se crea en el ***directorio actual*** un archivo binario llamado ***MatLab.mat*** con el estado del workspace (esto no incluye, por ejemplo, los gráficos, que deben guardarse separadamente). Dicho estado puede recuperarse la siguiente vez que se arranque el programa con el comando:

```
>> load
```

MATLAB permite varias opciones para almacenar las variables con las cuales se trabaja para su posterior utilización. En todos los casos el comando es **save.**

9.1.1.- Almacenamiento en código ASCII[5].

La sintaxis es:

```
save  <nombre del archivo>  <variables> -ascii
```

Ejemplo:

```
>> a = [0:1:100];
>> save ejemplo1.dat a -ascii
```

Esta forma de almacenar presenta la gran ventaja de que este archivo de datos puede ser leído por cualquier programa de manejo de texto y/o planillas. Por ejemplo: block de notas, Word, Excel, etc.

La terminación .dat no es obligatoria, pero se suele utilizar para identificar rápidamente el archivo como un archivo de datos en ascii.

El mayor inconveniente que tiene este método es que **todas las variables deben tener la misma dimensión** para ser almacenadas.

9.1.2.- Almacenamiento en binario

La sintaxis es:

```
save  <nombre del archivo>  <variables>
```

[5] En computación existen básicamente dos tipos de archivos, los archivos **ascii** y los archivos **binarios**. El vocablo **ascii** es un acrónimo para *American Standard Code for Information Interchange*. Es un estándar que asigna un valor numérico a cada carácter, con lo que se pueden representar los documentos llamados de Texto Plano, es decir, los que son legibles por seres humanos. Un archivo ascii es por tanto un archivos de texto, como los de extensión .txt (modificables con prácticamente cualquier editor de textos uno de formato de textos). Los archivos con extensión ***.m,*** son archivos de texto sin formato (archivos ASCII). Los archivos binarios son todos los demás.

Por defecto, **MATLAB** coloca a estos archivos la terminación **.mat.** Estos archivos no pueden ser leídos desde programas de procesamiento de texto, pero tienen algunas ventajas que se verán más adelante.

En este caso las variables almacenar pueden tener diferente dimensión.

Ejemplo:
```
>> a = [0:1:100];
>> b = sin(a);
>> save ejemplo2 a b
```

9.2.- Creación de un archivo desde el block de notas.

Los resultados experimentales que se obtendrán en el laboratorio serán procesados utilizando **MatLab**. Una opción para manejar estos datos desde el programa es generar desde el **block de notas** un archivo de datos. Este será un archivo ASCII. Para ello simplemente se debe abrir el **block de nota** e ingresar los datos en forma de columnas. Al almacenar tener la precaución de ponerle terminación **.dat** para identificarlos rápidamente como archivo de datos.

9.3.- Lectura de archivos de datos.

El comando para leer archivos de datos es *load***.** Veremos a continuación como trabajar en los dos formatos vistos anteriormente: ASCII y BINARIO. En todos los casos, la sintaxis de lectura es

```
>> load <nombre del archivo>
```

o alternativamente

```
>> load('nombre del archivo')
```

En ambos casos, el comando puede ejecutarse directamente, o ser asignado a una variable:

```
>> a=load('nombre del archivo')
```

9.4.- Archivos ascii.

Cabe notar que al leer datos de un archivo ASCII, la variable que tenemos en el workspace de MATLAB tiene el mismo nombre que el archivo, pero sin la terminación. Suele ser de utilidad asignar un nuevo nombre a esta variable:

```
b = datos1;
```

9.5.- Archivos binarios.

Los archivos binarios *.mat* contienen un estado del workspace con variables definidas de valores dados, como fuera guardado en algún momento. De esa manera, una vez leído el binario, las variables que tenemos en el workspace de **MATLAB** tiene el mismo nombre con que habían sido almacenadas, lo que representa una gran ventaja si se está realizando la lectura en el marco de un programa, puesto que se conoce a priori el nombre de las variables. Esto también sobrescribirá cualquier variable del mismo nombre que hubiera en el workspace antes de la lectura.

9.6.- Importación y exportación de datos

MATLAB es un software comúnmente utilizado para el análisis de datos experimentales que pueden provenir de distintas fuentes, incluidos otros programas. La forma de procesar estos datos externos es importarlos a MatLab.

De forma análoga, los datos calculados por MATLAB pueden ser transferidos o exportados a otras aplicaciones. Existen distintos tipos de datos (números, texto, audio, gráficos e imágenes), solo se describirá cómo importar y exportar datos numéricos.

La importación de datos se puede llevar a cabo mediante comandos o utilizando un Asistente de Importación de Datos (*Import Wizard*). Los comandos son útiles cuando el formato de los datos importados es conocido. MATLAB posee varios comandos que pueden ser utilizados para importar distintos tipos de datos. Los comandos de importación se pueden incluir en un archivo script, de forma que los datos se importen dentro del programa y antes de ser procesados por éste. El Asistente de Importación de Datos es útil cuando el formato de los datos (o el comando utilizado para la importación de datos) no es conocido. Este asistente determina el formato de los datos y posteriormente los importa automáticamente.

MATLAB permite la transferencia de datos directamente a formatos como *.csv* y **ASCII**, así como hojas de cálculo de tipo Lotus 123, etc.

Para llevar a cabo la importación de datos desde Excel se utiliza el comando *xlsread*. Este comando importa los datos de una hoja de cálculo Excel a una variable de tipo array. La forma más simple de utilizar este comando es:

```
nombre_variable = xlsread('nombre archivo')
```

'nombre archivo', introducido como cadena, es el nombre del archivo Excel. La ubicación de este archivo debe ser el directorio de trabajo actual o bien estar en la ruta de búsqueda. Si el archivo Excel importado tiene más de una hoja de cálculo sólo se importarán los datos de la primera de las hojas.

Si un archivo Excel contiene más de una hoja, se puede utilizar otra versión del comando *xlsread* para decidir cuál de ellas importar:

```
nombre_variable = xlsread('nombre archivo' ,'nombre hoja')
```

El nombre de la hoja debe introducirse como cadena.

Otra opción del mismo comando permite importar sólo una región (grupo de filas y columnas) de una hoja de cálculo determinada a partir de un archivo Excel:

```
nombre_variable = xlsread('nombre archivo' ,'nombre hoja', 'rango')
```

El *'rango'*, introducido como cadena, es una región rectangular de la hoja definida por la dirección (en notación Excel) de las celdas con respecto a las esquinas opuestas (superior izquierda, inferior derecha). Por ejemplo, 'C2:E5' representa una región de dimensión 4×3 a partir de las filas 2, 3, 4 y 5, y de las columnas C, D y E.

El paso inverso, es decir, la exportación de datos MATLAB a una hoja Excel, se lleva a cabo mediante el comando *xlswrite*, cuya sintaxis en su versión reducida es:

```
xlswrite('nombre archivo',nombre variable)
```

'*nombre archivo*', introducido como cadena, es el nombre del archivo Excel al cual se quieren exportar los datos. El archivo debe estar en el directorio actual. Si el archivo no existe se creará con el nombre especificado por parámetro.

nombre_ variable es el nombre de la variable MATLAB que contiene los datos que serán exportados.

Los argumentos '*nombre_hoja*' y '*rango*' también pueden ser añadidos al comando *xlswrite* para exportar los datos a una hoja concreta del archivo, dentro de un rango específico.

Ejemplo. La hoja de cálculo creada se llama DatosTest1, y se encuentra en la unidad de disco D. Una vez que el directorio por defecto se ha cambiado a la unidad D, se pueden importar los datos en MATLAB y asignárselos a la variable DATOS, como sigue:

```
>>DATOS= xlsread('DatosTest1')
DATOS=

  1.0000   2.0000   34.0000  14.0000
 15.0000   6.0000  -20.0000   8.0000
  3.0000  12.0000  -25.0000  -0.1000
 55.0000   9.0000   55.0000   9.0000
>>
```

10.- Creación de M-files

Si una tarea MATLAB la vamos a ejecutar muchas veces, es buena idea escribir un fichero con estos comandos para poder ejecutarlos tantas veces como queramos. Hasta ahora los comandos MATLAB que hemos visto se ejecutaban en la Ventana de Comandos. Aunque todos los comandos MATLAB se pueden ejecutar ele esta forma, la utilización de la Ventana de Comandos se restringe normalmente a la ejecución de un número pequeño de comandos con salidas bien controladas. En caso contrario, cuando el número de sentencias en demasiado elevada, es necesario plantearse la escritura y ejecución de código de otra forma. El problema proviene, básicamente, de que la Ventana de Comandos no es suficientemente interactiva, los comandos no pueden ser guardados y ejecutados de nuevo a petición del usuario. Eso implica que cada vez que se pulsa la tecla ***Intro*** sólo se ejecute el último comando, y todo lo anterior permanece inalterable, Si se necesita realizar alguna corrección o cambio sobre algunos de esos comandos previamente ejecutados, será necesario volver a escribirlos y ejecutarlos de nuevo de uno en uno.

Otra forma diferente de ejecutar comandos en MATLAB es crear un archivo con los comandos para ejecutarlo posteriormente. Cuando se ejecuta el archivo en cuestión, los comandos que contiene son ejecutados en el orden en que aparecen en el archivo. Además, si fueran necesarias correcciones o cambios posteriores, sólo habría que editar el archivo y ejecutarlo de nuevo.

MATLAB permite ejecutar secuencias de comandos almacenados en un archivo. Estos archivos deben tener la extensión "**.m**", y por eso se denominan **M-files.** Existen básicamente dos tipos de **M-files**: los denominados **function-files** y **script-files**.

Un fichero **.m** puede llamar a otros ficheros **.m** y ficheros de comandos pueden ser llamados desde ficheros de funciones. En estos casos es importante tener en cuenta la definición de las variables a utilizar, en la línea de que tengan un tratamiento local o global. Por defecto, MATLAB considera las variables locales, es decir, aunque varias funciones tengan la variable x, ésta es diferente en cada caso a no ser que haya sido definida como global.

Son ficheros de texto sin formato y que pueden crearse a partir de un editor de textos, no obstante, lo mejor es utilizar el editor del propio programa al que se accede por defecto al abrir un nuevo fichero.

La forma de editar M-files es usando un editor incorporado a **MatLab**, el denominado **MATLAB Editor/Debugger**, al cual se accede desde la opción '**File**' de la barra de menú, en la ventana de comando, cada vez que se abre un M-file (nuevo o ya existente), como se puede observar en la Figura 14.

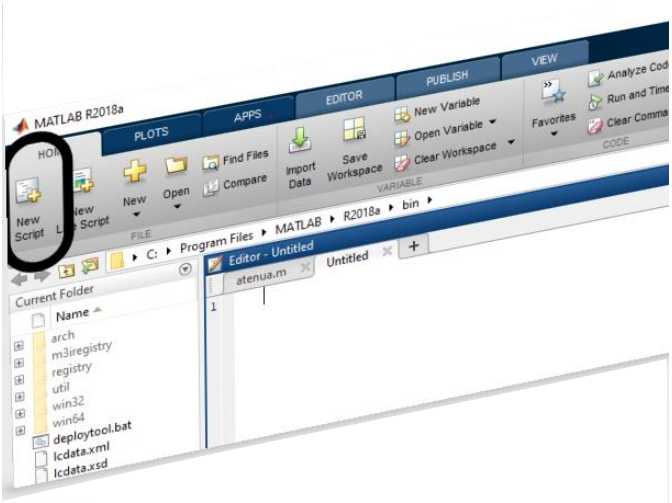

Figura 14**.** Ilustración de cómo crear un m-file

10.1.- Script-files[6]

Un script-file consiste de una sucesión de comandos **MATLAB** que se ejecutan sucesivamente cuando se teclea el nombre del archivo en la línea de comandos de MatLab. Es importante destacar que, en estos ficheros, las variables que se crean pertenecen al espacio base del Matlab, algo que no ocurre en los ficheros de función donde las variables pertenecen sólo al espacio de trabajo de esa función. Por ejemplo, si el archivo tiene el nombre **prueba.m**, cuando se lo corre desde la pantalla de **MATLAB (>>*prueba*)**, los comandos del archivo se ejecutan en el orden en que aparecen en el listado. Las variables en el script-file son globales. Esto significa que, si hay previamente en memoria alguna variable con el mismo nombre, su valor cambiará cuando se ejecute el programa. A su vez, todas las variables definidas dentro de un script quedarán en el workspace una vez terminado éste.

El primer paso es **abrir un editor de texto** para escribir el programa: desde la ventana de comandos de *MATLAB* se despliega el menú **FILE** y se elige la opción **new** m-file, a continuación, se desplegará la ventana del editor de *MATLAB* donde escribimos el programa.

Una vez terminado de escribir el *script*, se debe salvar (haciendo *clic* en *file* y luego haciendo *clic* en *guardar como*... aparecerá otra ventana donde elegimos un nombre para el archivo (npar.m por ej.) y el directorio donde guardar el archivo.

[6] También son conocidos por Ficheros de comandos o Programas propios del usuario.

Es importante recordar que los nombres de los programas en *MATLAB* deben tener extensión *.m* y que no deben contener espacios vacíos.

Un archivo script se puede ejecutar, bien tecleando su nombre en la Ventana de Comandos (y pulsando la tecla ***Enter***) o bien directamente desde la Ventana del Editor a través del icono ***Run*** (Ejecutar), Figura 15, o presionando la tecla F5.

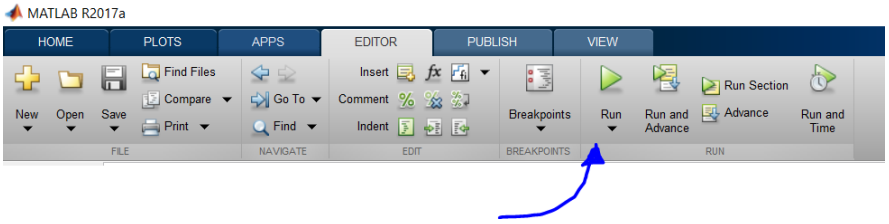

Figura 15 Icono Run para ejecutar un script.

10.2.- Programación con Matlab

La estructura general de un programa MATLAB es la siguiente:

1. Comentarios: inicialmente, pueden aparecer líneas comentadas en las que se da un título al programa y se realiza una breve descripción del mismo. Esta parte es opcional, pero es útil introducirla ya que se nos permite acceder directamente desde la ventana de comandos a la información comentada mediante la utilización del comando help, en la forma:

   >> help nombre del programa

2. Entrada de datos si se requiere: los datos necesarios para la resolución del problema deben suministrarse al programa mediante la lectura de sus valores por teclado o desde un fichero de datos.
3. Algoritmo: desarrollo de un procedimiento que permite obtener la solución del problema en función de los datos de entrada.
4. Salida de datos: los datos obtenidos como solución del algoritmo se deben ofrecer al usuario mediante escritura en pantalla o en un fichero de datos.

10.2.1.- Valores de entrada en un archivo script

Cuando se ejecuta un archivo script, las variables utilizadas en los cálculos dentro del archivo deben tener valores asignados previamente. La asignación de valores a estas variables se puede realizar de tres formas, dependiendo de dónde y cómo se haya definido la variable.

Variable definida y asignada en el archivo script

En este caso, la asignación del valor a la variable forma parte del archivo. Si el usuario quiere ejecutar el archivo con un valor diferente para dicha variable, el archivo debe ser editado para asignar un nuevo valor a la variable. Una vez que el archivo se ha guardado en disco, se puede ejecutar de nuevo.

El siguiente es un ejemplo de un caso de este tipo. El archivo script (guardado en disco como ***Jugada.m***) calcula la puntuación media de tres jugadas distintas.

```
% Este archivo script calcula la puntuación media de tres jugadas distintas.
% La asignación de las variables que contienen los puntos es parte del archivo script.
jugada1 = 75;
jugada2 = 93;
jugada3 = 68;
puntuacíon_media = jugadal + jugada2 + jugada3)/3
```

Una vez ejecutado el archivo, la Ventana de Comandos se verá:

```
>>Jugada
puntuacion_media =
78.6667
```

Variable definida y asignada en la Ventana de Comandos

En este caso, la asignación de un valor a la variable se realiza en la Ventana de Comandos. Si el usuario quiere ejecutar el archivo script con un valor diferente para esta variable, se debe asignar el nuevo valor en la Ventana de Comandos, y después ejecutar el archivo de nuevo. En el ejemplo anterior, en el cual el script consistía en un programa que calculaba la puntuación media de tres jugadas, el archivo script almacenado para este caso (guardado como ***Jugada***) sería:

```
% Este archivo script calcula la puntuación media de tres jugadas distintas.
% La asignación de las variables que contienen. los puntos (jugada 1, jugada2, jugada3)
% se lleva a cabo en la Ventana de Comandos.
puntuación_media=(jugada1+jugada2+jugada3)/3
```

Una vez ejecutado el archivo (con reasignación de valores), la Ventana de Comandos se verá:

```
>>jugada1 = 75;
>>jugada2 = 93;
>>jugada3 = 68;
>>Jugada
puntuacion_media =
78.6667
>>
>>jugada1 = 87;
>>jugada2 = 70;
>>jugada3 = 50;
>>Jugada
puntuacion_media =
69
>>
```

Variable definida y asignada en el archivo script, pero además se introduce un valor concreto para la variable cuando se ejecuta el archivo en la Ventana de Comandos

En este caso, la variable se define en el archivo script y cuando se ejecuta dicho archivo al usuario se le pide un valor concreto, a través de la Ventana de Comandos, para asignárselo a la variable del archivo script. Para hacer esto se utiliza el comando ***input***. La forma de utilizar este comando es la siguiente:

```
nombre_variable = input ('Mensaje que se muestra en la Ventana de Comandos')
```

Cuando se ejecuta el comando ***input*** como parte del script, la cadena que va entre paréntesis se visualiza en la Ventana de Comandos. La cadena simboliza un mensaje a partir del cual se le pedirá al usuario un valor para la variable ***nombre_variable***. Introducido el valor y pulsada la tecla ***Intro***, la variable será asignada con el valor que el usuario haya tecleado. Al igual que sucede con otras variables, el comando ***input*** deberá finalizarse con punto y coma para que el valor de la variable no se muestre en la Ventana de Comandos. A continuación, se muestra un archivo script que utiliza el comando input para introducir los puntos de las distintas jugadas, a partir del ejemplo anterior.

```
% Este archivo script calcula la puntuacion media de tres jugadas distintas.
% La asignación dé las variables que contienen los puntos (jugada1, jugada2, jugada3)
% se lleva a cabo mediante el comando input.

jugada1 = input('Introduzca la puntuacion de la primera jugada');
jugada2 = input('Introduzca la puntuacion de la segunda jugada');
jugada3 = input('Introduzca la puntuacion de 1a tercera jugada');
puntuacíon_media = (jugadal + jugada2 + jugada3)/3
```

Una vez ejecutado el archivo, la Ventana de Comandos se verá (el archivo script ha sido guardado con el nombre de Jugada):

```
>>Jugada
Introduzca la puntuacion de la Primera jugada 67
Introduzca la puntuacion de la segunda jugada 91
Introduzca la puntuacion de la tercera jugada 70
puntuacion_media =
76
>>
```

En el ejemplo anterior MATLAB muestra los mensajes de los comandos ***input***. El usuario introduce cada uno de estos valores pulsando, al final de ellos, la tecla ***Intro***.

En este ejemplo se asignan valores escalares a las variables, pero se podrían asignar valores a vectores y matrices, asignando los valores entre corchetes.

El comando ***input*** también se puede utilizar para asignar una cadena a una variable. Esto se puede realizar de dos formas. Una es utilizar el comando como ya se ha venido mostrando, introduciendo en este caso la cadena entre comillas simples una vez que el mensaje del comando input se visualiza en la Ventana de Comandos. Otra forma

es utilizar una opción del comando ***input*** que permite concretar que los caracteres serán introducidos como cadena Este comando tiene la siguiente forma:

```
nombre_variable = input('Mensaje' ,'s')
```

donde 's', dentro del comando, indica que se introducirán caracteres en la entrada. En este caso, cuando el mensaje aparece, la cadena se puede introducir sin comillas simples, pero es asignada a la variable como una cadena. Un ejemplo de utilización de input con esta opción sería (nombre del script ConversionEnergia):

```
% Este archivo script convierte la unidad de la energía
Ein = input('Introduzca el valor de la energia trabajo que hay que convertir: ');
EinUnits = input('Introduzca la unidad actual de la energía (J, ft-lb, cal, eV): ','s');
EoutUnits = input('Introduzca la nueva unidad para la energia-(J, ft-lb, cal, eV): ','s');
……
sentencias
……
```

Una vez ejecutado el archivo, la Ventana de Comandos se verá

```
>>ConversionEnergia
Introduzca el valor de la energia (trabajo) que hay que convertir: 2800
Introduzca la unidad actual de la energia (J, ft..:lb, cal, eV): cal
Introduzca la nueva unidad para la energia (J, ft-lb, cal, eV): J
E= 11715.5 J
>>
```

10.3.- Function-files

Los *function-files* permiten extender la biblioteca de funciones **MATLAB** para el uso en aplicaciones específicas. Una función permite escribir un fragmento de código parametrizado. De esta forma, es posible escribir un bloque de código y ejecutarlo para distintos datos. Otra ventaja de las funciones es que permiten estructurar u organizar el código de un programa. Un problema complejo se puede resolver mediante un programa extenso o, es mejor descomponerlo en sub-problemas de complejidad moderada para ser resueltos por sub-programas sencillos, como son las funciones. En lugar de utilizar un programa muy grande para resolver un problema complejo se emplean distintos subprogramas que resuelven tareas sencillas y que se combinan para producir una solución final más simple. Hasta ahora hemos utilizado muchas funciones internas de MatLab, como max o sum. En este tema aprenderemos a escribir nuestras propias funciones.

A diferencia de los *script-files*, las variables en un *function-file* son locales, significando esto que las funciones generan un 'workspace virtual' al ser llamadas, con sus propias variables. Por lo tanto, ninguna variable del mismo nombre que ya existiera en el workspace del usuario será modificada por la función, y ninguna variable definida dentro de la función quedará en el workspace una vez que ésta termine. En ambos casos, la excepción es la variable

(o argumento) de salida de la función. *function* es una palabra reservada con la que hay que iniciar la definición. La primera línea de la definición de una función se llama cabecera de la función y contiene la interfaz de uso de la función. Si una función está bien documentada, basta con conocer su cabecera y su documentación para poder utilizarla sin tener que leer su código. En. este sentido una función proporciona un mecanismo de encapsulación de código que nos permite abstraernos de los detalles de implementación.

La primera línea de un *function-file* debe comenzar con la palabra *function* seguida de una expresión donde se declara el nombre de la función y se indica cómo la función debe ser llamada desde el espacio de trabajo de **MatLab**, y cuáles son los argumentos de entrada y salida de la función. El nombre del *function-file* debe ser de la forma *function-name.m*.

El formato de una función es la siguiente

```
function [vr1 vr2 . . . vrm] = nombre (param1, param2, . . . , paramn )
% documentacion
bloque de codigo
end
```

[vr1 vr2 ... vrm] son las variables de retorno de la función, también llamados parámetros de salida. Se utiliza una sintaxis similar a la de creación de un vector, pero representan los valores que devuelve la función.

(param1, param2, ..., paramn) es una lista de nombres de variables encerrada entre paréntesis y separada por comas. Representa los parámetros formales de la función, que en MATLAB se corresponde con los parámetros de entrada de la función.

end el bloque de código termina con la palabra *end*, aunque si la función aparece en un archivo que contiene únicamente a esa función el *end* no es necesario (se recomienda siempre su uso).

En la Figura 16 se muestra el ejemplo de crear una función que suma dos variables Abrimos un archivo **M** y lo guardamos en nuestra carpeta.

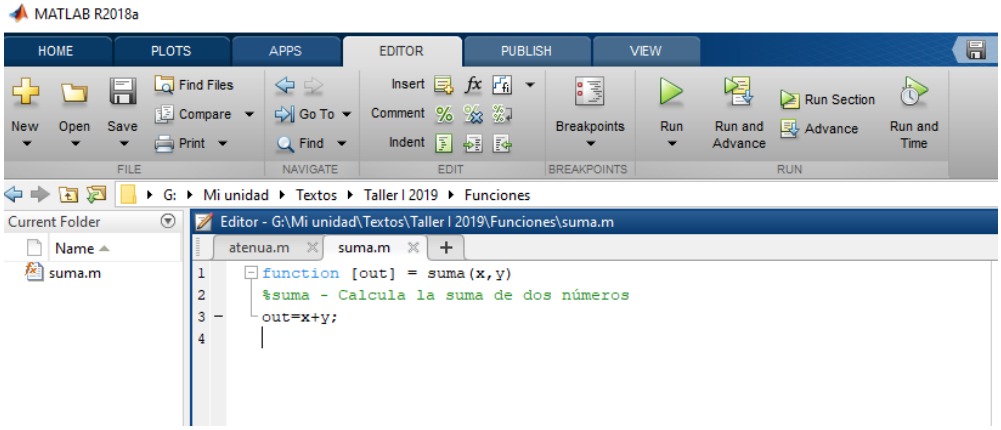

Figura 16. Función que suma dos variables

```
function [out] = suma(x,y)
%suma – Calcula la suma de dos números
out=x+y;
```

Ahora se introduce

```
>> s32 = suma(2,3)
S32 =
5
```

Las líneas que comienzan con el carácter '%' corresponden a comentarios. Las líneas de comentarios que están inmediatamente debajo de la primera línea pueden utilizarse para aprovechar las facilidades del *help* de **MatLab**. El texto correspondiente a esas líneas aparecerá en el espacio de trabajo cuando se tipee *help function-name* y por lo tanto las mismas se utilizan generalmente para describir la operación que realiza la función. Es recomendable incluir *siempre* esta documentación en un *function-file*.

```
>> help suma
suma – Calcula la suma de dos números
```

Los comentarios pueden ser utilizados para explicar cualquier línea de un m-file.

Funciones auxiliares

Un archivo **m** puede contener varias funciones. Sin embargo, sólo la primera de ellas puede ser invocada desde fuera del archivo, desde la ventana de comando o desde otra función ubicada en otro archivo. ¿Cuál es la utilidad entonces de escribir varias funciones en un archivo **m**? La respuesta es sencilla, a veces una función es lo suficientemente compleja como para que amerite dividir su código en varias funciones. A la primera función ubicada en un archivo **m** se le llama *función principal* y al resto *funciones auxiliares*. Las funciones auxiliares sólo pueden ser invocadas desde otras funciones ubicadas en el mismo archivo **m**.

Ejemplos

E.3) Vuelva a hacer en un m-file los ejemplos 1 y 2 agregando las siguientes partes en cada uno de ellos

a) Almacene en un archivo binario todas las variables definidas anteriormente.

b) Luego borre las variables

c) Lea el archivo almacenado en el ítem a) y ejecute el comando *whos*.

E.5) Escriba una función cuya entrada sean dos coordenadas en dos dimensiones (x, y), y que devuelva su correspondiente radio y Angulo. Recuerde que el radio se define como $r = \sqrt{x^2 + y^2}$ y el ángulo según $tan\theta = \frac{y}{x}$

Para poder utilizar una función en MATLAB , ésta debe estar almacenada en un archivo de texto con extensión *.m*. Lo normal es ponerle al archivo el mismo nombre que la función, pero con la extensión *.m*. Por ejemplo, para

la función de la Figura 16 el archivo se llamará *suma.m.* Si se usa nombres distintos para nombrar a la función y al archivo es el nombre del archivo y no el de la función el que hay que usar para invocar a la función (esta opción no es aconsejable). Para que se pueda invocar a la función el archivo debe estar almacenado en la carpeta o directorio de trabajo.

11.- Gráficas

Los gráficos son herramientas muy utilizadas para presentar todo tipo de información; información que puede proceder de cualquier campo del conocimiento, pero especialmente de las disciplinas relacionadas con las ciencias y la ingeniería, donde MATLAB es ampliamente utilizado. Las facilidades para la generación de gráficos y el cálculo con matrices son dos de las características más destacables de MatLab. Con los comandos de MATLAB se pueden crear distintos tipos de gráficos: estándares con ejes lineales, logarítmicos o semilogarítmicos, de barras, polares, de malla y de superficies de contorno tridimensional, etc. Estos gráficos se pueden personalizar para que tengan la apariencia deseada, Así, se puede establecer el tipo, el color y el grosor de línea; se pueden añadir líneas de referencia y cuadrículas; y también títulos y comentarios. Además, se pueden superponer varios gráficos sobre un mismo sistema de ejes de coordenadas, o poner varios gráficos en una misma página. Cuando un gráfico tiene varios tipos de datos, también se pueden añadir leyendas. En resumen, MATLAB tiene un excelente manejo de gráficos. Aquí veremos sólo algunos de los comandos básicos para la generación de gráficas en 2 dimensiones. Para aquellos estudiantes que tengan interés, recomendamos hacer un help graphics, help graph2d y help graph3d.

Lista de algunos comandos:

plot Crea el gráfico

hold Permite realizar la superposición de 2 o más gráficos en una misma pantalla. Es un comando on/off.

figure Genera una nueva pantalla de gráficos.

11.1.- La función plot

La función más usada para crear gráficos en 2D es *plot*. Esta función es bastante versátil El comando más simple es *plot(x, y, cad)*, que utiliza dos vectores(arrays unidimensionales), *x* e *y*, de la misma longitud y ***cad*** es una cadena de caracteres que especifica el modo de visualizar las líneas que unen los datos y los propios datos. Éste dibujará los puntos (x_i, y_i) y los unirá mediante rectas continuas. Si se omite el parámetro *x* entonces las coordenadas $\boldsymbol{x}$ son los índices del vector $\boldsymbol{y}$, es decir, *1:length(y)*. Cuando se ejecuta el comando *plot*, el gráfico se crea en la Ventana de Gráficos. Si no se ha abierto previamente, esta ventana se abrirá automáticamente al ejecutar el comando. La representación del gráfico que se mostrará se corresponde con una curva donde los valores de **x** serán los de la abscisa (eje horizontal) y los de **y** los de la ordenada (eje vertical). La curva se construye mediante segmentos de recta que unen los puntos cuyas coordenadas están definidas por los elementos de los vectores *x* e *y*. El vector que se introduce como primer argumento de *plot* será el que defina el eje horizontal, mientras que el segundo definirá el eje vertical.

Para personalizar los gráficos, si se desea, el comando *plot* admite en su sintaxis otros argumentos que se pueden utilizar para definir el color y estilo de líneas y marcadores. Al utilizar estas opciones, el comando *plot* amplía su sintaxis:

plot(x,y, 'especificadores de linea', 'Propiedades', 'Valores')

(Opcional) Especificadores que definen el tipo y color de líneas y marcadores.

(Opcional) Propiedades, junto con sus valores, que se pueden utilizar para especificar el grosor de la línea y el tamaño de los marcadores, así

Los especificadores se introducen como cadenas de caracteres dentro del comando *plot*. Dentro de 1a cadena, los especificadores se pueden teclear en cualquier orden. Los especificadores son opcionales. Esto significa que un comando puede contener uno, dos o tres especificadores, o bien ninguno.

Si no se le da ningún vector *x*, MATLAB asume que $x(i) = i$. A continuación *plot(y)* recibe el mismo espacio en el eje de las *x*: los puntos son $(i, y(i))$.

Se pueden cambiar el tipo y color de la línea que une los puntos mediante un tercer argumento. Si este argumento no existe, MATLAB dibuja por defecto una línea continua de color azul "**-**".

Introduciendo *help plot* se obtienen muchas opciones, aquí sólo indicamos unas pocas a modo de ejemplo:

plot(x, y, 'r+') dibuja los puntos en forma de +, y en rojo.

plot(x, y,' --') dibuja una línea discontinua y plot(x, y,'·'), una línea de puntos.

Se pueden omitir las líneas y representar sólo los puntos discretos de distintas formas:

plot(x, y,' o') dibuja círculos. Otras opciones son '+', 'x' o '*'.

A continuación, listamos los posibles colores, tipos de línea y puntos. El formato por defecto es *b*-, es decir, azul, con línea continua y sin resaltar los puntos. Comencemos por los colores:

b blue	m magenta
c cyan	r red
g green	w white
k black	y yellow

Se puede especificar el carácter o el nombre completo del color. Listamos ahora los tipos de puntos:

o círculo	**p** estrella de cinco puntas (pentagram)
d diamante o rombo	+ más
h estrella de seis puntas (hexagram)	**.** punto

	< triángulo hacia la izquierda
***** asterisco	> triángulo hacia la derecha
v triangulo hacia abajo	**x** x
^ triángulo hacia arriba	**s** cuadrado

Los tipos de línea se especifican con los caracteres:

- - discontinua con rayas
-. discontinua con guiones y puntos
: discontinua con puntos
- continua

A modo de ejemplo el comando:

plot(x,y,'-mo' ,'LineWidth' ,2,'markersize',12,'MarkerEdgeColor','g','markerfacecolor','y')

crea un gráfico en el cual una línea sólida ***(-)*** de color magenta (***m***) une puntos que se representan mediante marcadores en forma de círculo (***o***). El grosor de línea es de dos puntos, y el tamaño de los círculos utilizados como marcadores es de 12 puntos. Además, los marcadores (círculos) tienen bordes de color verde, y amarillo como color de relleno.

Para obtener dos gráficas en los mismos ejes, utilizar *plot(x, y, X, Y)***.** Sustituyendo *plot* por *semilogx*, *semilogy*, o *loglog* se cambian uno o ambos ejes respectivamente a la escala logarítmica.
El comando *axis([a b c d])* ajusta el tamaño del gráfico al del rectángulo a ≤ x ≤ b, c ≤ y ≤ d. Para dar título al gráfico o marcar los ejes de las x o de las y, se escribe entre comillas la etiqueta deseada, como en los ejemplos siguientes:

title (‘altura del satélite’)
xlabel (‘tiempo en segundos’)
ylabel (‘altura en metros')

El comando *hold* sirve para superponer dos o más gráficos. Este comando alterna entre las opciones *on* y *off*. Para imprimir o guardar la pantalla de gráficos en un archivo, véase *help print*.

Con las nuevas versiones de MatLab, todo el etiquetado de los ejes, texto, título, etc. de las gráficas se puede controlar desde la propia pantalla de gráficos. Esto se verá con más detalle en los ejercicios.

Ejemplo

E6) Defina un vector t (cuyo primer elemento es 0; el último es 512; y el paso es 0.1) y un vector x = 3t^2+2t-3 y un vector z = exp(t)

a) Grafique x en función de t. Coloque nombre a los ejes, cambie los símbolos, color, coloque título, grilla, etc.

b) Ídem para z en función de t, pero en otra ventana de gráfica

c) Grafique x(t) y z(t) en la misma gráfica.

d) Grafique en la misma ventana, pero en distintas gráficas x(t) y z(t)

f) Investigue el comando ginput.

11.2.- Gráficos paramétricos

Un gráfico no tiene que restringirse a una variable dependiente, normalmente sobre el eje y, que depende de una variable independiente visualizada en el eje x. En un gráfico paramétrico las variables en los distintos ejes dependen de una variable independiente. Esa variable independiente define un camino sobre la superficie de dibujo. Por ejemplo, para dibujar una circunferencia de radio 2 se puede ejecutar:

```
>>radianes = linspace ( 0 , 2 * pi , 4 0 ) ;
>>radio = 2;
>>plot (cos (radianes) * radio, sin (radianes) * radio )
>>axis ('equal')
```

11.3.- Entrada gráfica

La función ***ginput*** permite capturar las coordenadas de un número ilimitado de puntos de la figura activa utilizando el ratón o las flechas de desplazamiento. Prueba a ejecutar:

```
>> x = - 2 : 2 ;
>> plot ( x , x . ˆ 2 , ' r * ' )
>> [ x y ] = ginput
x =
-1.2949
0.0323
0.6129
y =
2.9532
1.7836
3.1637
```

Se puede seleccionar puntos pulsando el ratón. Las coordenadas de los puntos seleccionados se guardan en $x(i);\ y(i)$. Pulsando la tecla ***Enter*** se termina la lectura de puntos. También es posible la sintaxis ***ginput(n)*** para leer ***n*** puntos.

11.4.- Generación de gráficos a partir de datos

En este caso, primero se crean vectores con los datos, y luego se utilizan estos vectores en el comando plot para generar el gráfico. He aquí un ejemplo de representación gráfica de los datos correspondientes a ventas de una compañía desde el año 1988 a 1994.

año	1988	1989	1990	1991	1992	1993	1994
Venta (millones)	8	12	20	22	18	24	27

Para representar estos datos, el listado de años se asigna a un vector (llamado **an**) y las ventas a un segundo vector (llamado **ven**). A continuación, se muestra la Ventana de Comandos donde se crean los vectores y se utiliza el comando ***plot***:

```
>>an = [1988:1:1994];
>>ven= [8 12 20 22 18 24 27];
>>plot(an,ven,'--r*' ,'linewidth' ,2,'matkersize' ,12)
```

Especificadores de línea: línea discontinua con marcadores rojos en forma de asterisco.

Propiedad y valor: el ancho de línea es de 2 puntos y el tamaño del marcador (asteriscos) es de l2 puntos.

11.5.- Generación de gráficos a partir de funciones

En muchos casos es necesario representar un gráfico a partir de una determinada función matemática. Esto se puede hacer utilizando los comandos ***plot*** y ***fplot***.

Para representar una función del tipo $y = f(x)$ con el comando ***plot***, el usuario necesita crear primero un vector con los valores de x del dominio de la función que se va a representar. Seguidamente deberá crear un vector y con los correspondientes valores de $f(x)$, utilizando para ello operaciones elemento a elemento. Una vez creados ambos vectores, se puede utilizar el comando plot para representar la función.

Ejemplo, vamos a utilizar el comando ***plot*** para representar gráficamente la función $y = 3{,}5^{-0{,}5x}\cos(6x)$ en el intervalo $-2 \leq x \leq 4$. Un programa que genera el gráfico de esta función se muestra en el siguiente archivo script (Figura 17).

```
% Fichero script que genera el grafico de
% la funcion: 3.5^(-0.5*x).*cos(6*x)
x= -2:0.01:4;
y=3.5.^(-0.5*x).*cos(6*x);
plot(x,y)
```

Figura 17 Ejemplo de aplicación de la función plot

MATLAB representa las funciones mediante segmentos de líneas rectas que conectan cada uno de los puntos que resultan de la función para cada uno de los valores del dominio. Para obtener una resolución adecuada en la representación gráfica, los elementos del vector x deben ser apropiados. Cuánto más rápidamente varíe una función, menor deberá ser el espaciado entre los valores del dominio. En el ejemplo anterior, el valor del espaciado o distancia entre los elementos es 0,01, y produce el gráfico que se muestra en la Figura 18. Sin embargo, si representamos esa función con el mismo dominio, pero con un espaciado mucho mayor (por ejemplo, de 0,6), la gráfica que se obtiene, mostrada en la Figura 19, aparece distorsionada. Se trata de la misma función, pero con menor resolución.

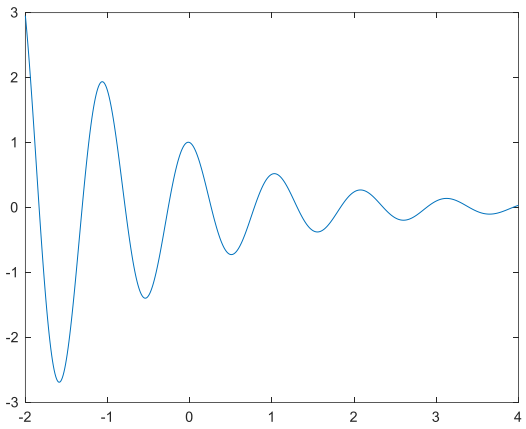

Figura 18 Gráfico de la función $y = 3{,}5^{-0{,}5x}\cos(6x)$

El comando ***fplot*** representa gráficamente una función de la forma y = f(x) entre unos límites especificados por el usuario. El comando tiene la siguiente sintaxis:

fplot ('funcion', limites, especificadores de linea)

funcion: La función se puede teclear directamente como cadena dentro del comando. Por ejemplo, si la función que se quiere representar es $f(x) = 8x^2 + 5\cos(x)$, esta función se puede introducir en forma de cadena como: '8*x^2+5*cos(x)'

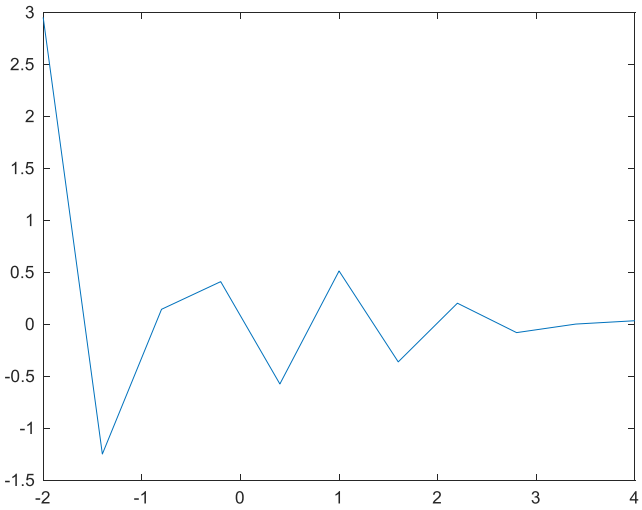

Figura 19 Gráfico de la función $y = 3{,}5^{-0{,}5x}\cos(6x)$ con un espaciado mayor.

- La función que se va a representar se puede construir dentro de la cadena utilizando cualquier letra como variable. Por ejemplo, la función anterior se puede teclear como '8*z^2+5*cos(z)' o como '8*t^2+5*cos(t)'.

limites: Los límites se especifican mediante un vector de dos elementos [xmin, xmax] que define el dominio de la variable x, o también mediante un vector de cuatro elementos [xmin, xmax, ymin, ymax] que define los dominios para los ejes x e y, de dos en dos respectivamente.

especificaciones de linea: Funcionan igual que en el comando *plot*. Como ejemplo, para representar una función del tipo $y = x^2 + 4sen(2x) - 1$, para $-3 \leq x \leq 3$, se puede introducir el siguiente comando en la Ventana de Comandos

```
>> fplot ('x^2+4*sin(2*x)-1',[-3 3])
```

La Figura 20 muestra el resultado que se genera.

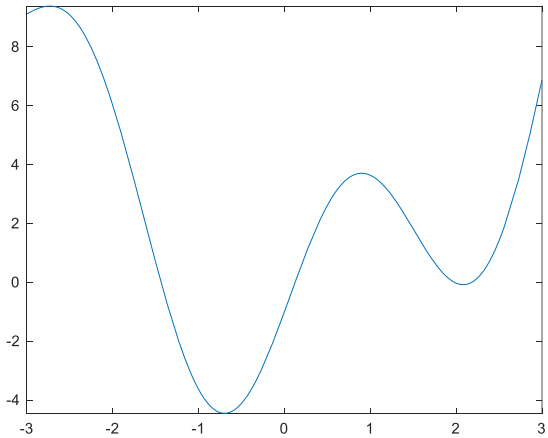

Figura 20 representar una función de $y = x^2 + 4sen(2x) - 1$ para $-3 \leq x \leq 3$

En las versiones más actuales la sentencia es fplot(@(x)x.^2+4.*sin(2.*x)-1, [-3,3]).

11.6.- Representación gráfica de varias funciones a la vez

A veces resulta interesante representar varias funciones a la vez. La representación de varias funciones se puede realizar de tres formas diferentes. Una de ellas consiste en utilizar el comando ***plot***, otra en utilizar los comandos ***hold on*** y ***hold off***, y la tercera consiste en utilizar el comando ***line***.

Supongamos que se quiere representar gráficamente tres funciones: $y(x)$, $v(u)$ y $t(h)$. En el caso del comando ***plot*** , su sintaxis es , tecleando las funciones que se van a representar como pares de vectores, de la forma:

plot(x, y, v, u, t, h)

La forma de representar varias funciones en un mismo gráfico con ***hold***, es utilizar primero el comando ***plot*** para representar la primera función, y luego introducir el comando ***hold on***. Este comando mantiene la Ventana de Gráficos con el primer gráfico abierto, conservando los mismos ejes y el formato establecido. Una vez introducido este comando se proceden a ejecutar tantos comandos *plot* como se quieran. Finalmente se introduce o ejecuta el comando ***hold off*** para decirle al sistema que no se desean más representaciones sobre la misma región gráfica (el comando plot vuelve al estado por defecto).

Ejemplo:

```
>> x= [-2:0.01:4];
>> y=3*x.^3-26*x+6;
>> yd=9*x.^2-26;
>> ydd=18*x;
>> plot(x,y)
>> hold on
>> plot(x,yd,'--')
>> plot(x,ydd,'-.')
>> hold off
```

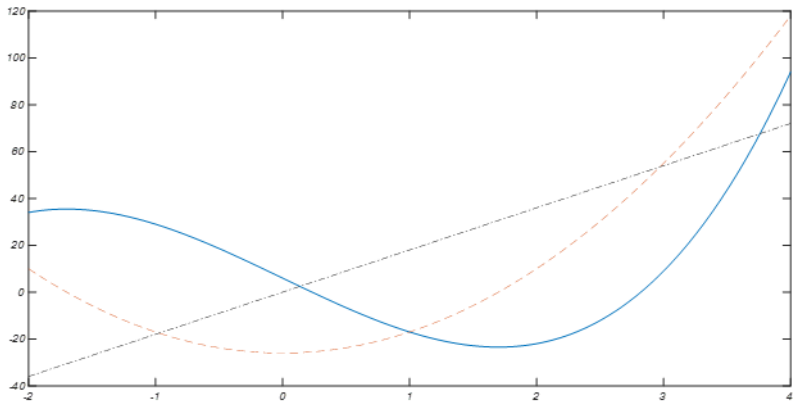

Figura 21 Ejemplo de utilización de los comandos hold para una función y sus derivadas

Con el comando *line* se pueden añadir curvas (líneas) adicionales a un gráfico que ya existe. La sintaxis del comando es la siguiente:

line(x,y,Propiedades,Valores)

Las propiedades, junto con sus valores, se pueden utilizar para especificar el estilo, el color y el grueso de la línea, así como el tipo, el tamaño y los colores de borde y relleno de los marcadores.

La sintaxis del comando *line* es muy parecida a la del comando *plot*, aunque este comando no tiene especificadores de línea. El estilo, el color y los marcadores utilizados se deben especificar mediante las ya conocidas propiedades y sus correspondientes valores. Estas propiedades son opcionales; si no se introduce ninguna, MATLAB asigna propiedades por defecto. Por ejemplo, el comando:

line(x,y, 'linestyle', '--','color',' r', 'marker', 'o')

crearía, en un gráfico existente, una nueva curva discontinua con marcas circulares.

La diferencia fundamental entre *plot* y *line* es que el comando *plot* crea un nuevo gráfico cuando se ejecuta, mientras que el comando *line* simplemente añade una nueva curva a uno que ya existe previamente. Para crear una representación de varias funciones (Figura 22), la primera se crea con un comando *plot*, y seguidamente se introducen tantos comandos 1ine como funciones adicionales se desee (si se introduce un comando line antes que uno *plot* el sistema dará un error).

```
>> x= [-2:0.01:4];
>> y=3*x.^3-26*x+6;
>> yd=9*x.^2-26;
>> ydd=18*x;
>> plot(x,y)

>> line(x,yd,'LineStyle','--','color','r')
>> line(x,ydd,'LineStyle','-.','color','k')
```

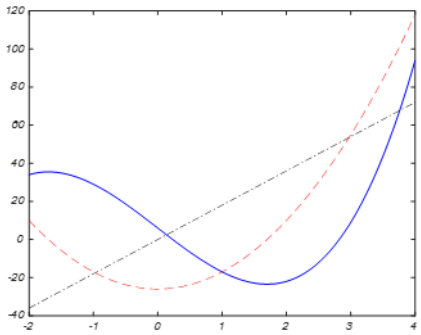

Figura 22 Ejemplo de utilización de los comandos line para una función y sus derivadas

11.7.- Formateado de una representación gráfica

Los comandos *plot* y *fplot* crean representaciones gráficas de apariencia muy sencilla y es necesario formatearlas para que tengan un determinado aspecto y añadirles información propia del gráfico como, por ejemplo, el título del gráfico, etiquetas y escalas personalizadas en los ejes de coordenadas, leyendas, cuadrículas, etiquetas de texto, etc.

Los gráficos se pueden formatear utilizando comandos de MATLAB a continuación de los comandos *plot* o *fplot*, o utilizando el editor de gráficos de la Ventana de Gráficos. El primer método es útil cuando los comandos *plot* o *fplot* son parte de un programa (archivo script). Por otra parte, el formato que se aplica mediante el editor de gráficos de la Ventana de Gráficos sólo se mantiene para un gráfico en concreto, y se tendrá que volver a aplicar la próxima vez que se cree el gráfico.

Los comandos de formato se introducen después de aquellos que crean o visualizan gráficos: plot y fplot. Existen varios comandos de este tipo, entre ellos se encuentran:

- **xlabel, ylabel**: sirven para poner un título, en forma de texto. Su sintaxis es: xlabel(‘texto’); ylabel(‘texto’).
- **title**: añade un título (principal) al gráfico, en la parte superior del mismo. Su sintaxis es: title(‘texto’).
- **text**: permite situar una etiqueta de texto dentro del gráfico. El comando admite dos variantes: text(x,y,'texto') y gtext ('texto'). El comando ***text*** coloca el texto en el gráfico de manera que el primer carácter se sitúe en el punto con coordenadas ***x*** e ***y*** (según los ejes del gráfico). En cambio, el comando ***gtext*** coloca el texto en la posición especificada por el usuario. Cuando se ejecuta este comando, se abre la Ventana de Gráficos y el usuario especifica la posición pulsando con el ratón en el punto deseado

- **legend**: coloca una leyenda en la representación gráfica. Las leyendas incluyen una muestra del tipo de línea de cada función que se representa y una etiqueta especificada por el usuario, que permite indicar a qué corresponde cada muestra. La sintaxis de este comando es la siguiente: legend ('cadenal', 'cadena2', ... , posicion). Las cadenas son las etiquetas que se colocan junto a las muestras de línea, y su orden debe corresponderse con el orden en el cual se han introducido las funciones. La variable posicion es un número opcional que especifica el sitio en el que se situará la leyenda dentro del gráfico. Los valores posibles son:
-

posición=-1	Sitúa la leyenda fuera de los límites establecidos por los ejes del gráfico, en el lado derecho.
posición=-0	Sitúa la leyenda dentro de los límites establecidos por los ejes del gráfico en una posición que interfiera lo menos posible con el gráfico.
posición=1	Sitúa la leyenda en la esquina superior derecha del gráfico (opción por defecto).
posición=2	Sitúa la leyenda en la esquina superior izquierda del gráfico.
posición=3	Sitúa la leyenda en la esquina inferior izquierda del gráfico.
posición=4	Sitúa la leyenda en la esquina inferior derecha del gráfico.

- **axis**: Cuando el comando plot(x,y) se ejecuta, MATLAB crea los ejes correspondientes para la representación gráfica, basándose en los valores máximo y mínimo de los valores posibles que toman **x** e **y**. El comando axis permite cambiar el rango de los ejes, así como su apariencia. A continuación, se muestran algunos de los posibles formatos que acepta el comando axis

axis([xmin, xmax, ymin, ymax])	Establece los límites de ambos ejes, x e y, entre los valores máximos y mínimos a partir de los valores xmin, xmax, ymin, ymax.
axis equal	Establece la misma escala en ambos ejes.
axis square	Establece la región de los ejes en un cuadrado.
axis tight	Establece los límites de los ejes en función del rango de los datos.

- **grid**: *grid on*, añade una cuadrícula a la representación gráfica. *grid off*, elimina la cuadrícula de la representación gráfica.

11.8.- Varios gráficos en una figura: subplot

La función *subplot* permite visualizar varios gráficos en diferentes zonas de una figura. Con la sintaxis *subplot(fila,col ,n)* se divide la figura actual en una matriz con *filax-coly* áreas de dibujo y se establece la n-ésima como la actual para dibujar. La numeración de las áreas va de la fila superior a la inferior y dentro de cada fila de la izquierda a la derecha. La Figura 23 muestra la numeración de los subplot para el caso de 2 filas y 2 columnas

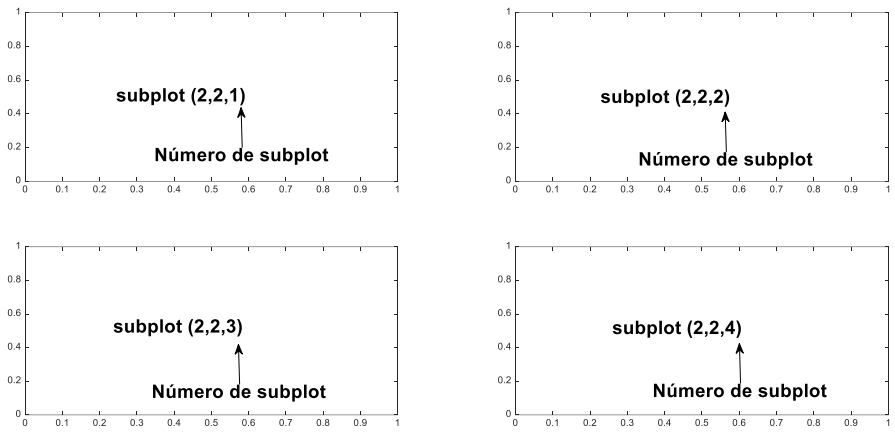

Figura 23 Numeración de los subplot, caso de 2×2

Como ejemplo, construyamos la siguiente figura

```
subplot (2, 2, 1 )
plot (1:100 )
title ('x')
subplot (2,2,2)
plot (sqrt (1:100))
title ('sqrt ( x )')
subplot (2,2,3)
plot (log(1:100))
title (' logartimo neperiano ')
subplot (2,2, 4)
x = 0:0.1:8 *pi;
plot (x, sin(x).* x)
title (' seno con amplitud variante ')
```

donde se divide la figura en una matriz 2x2, donde se dibujan distintos gráficos, obteniéndose la Figura 24.

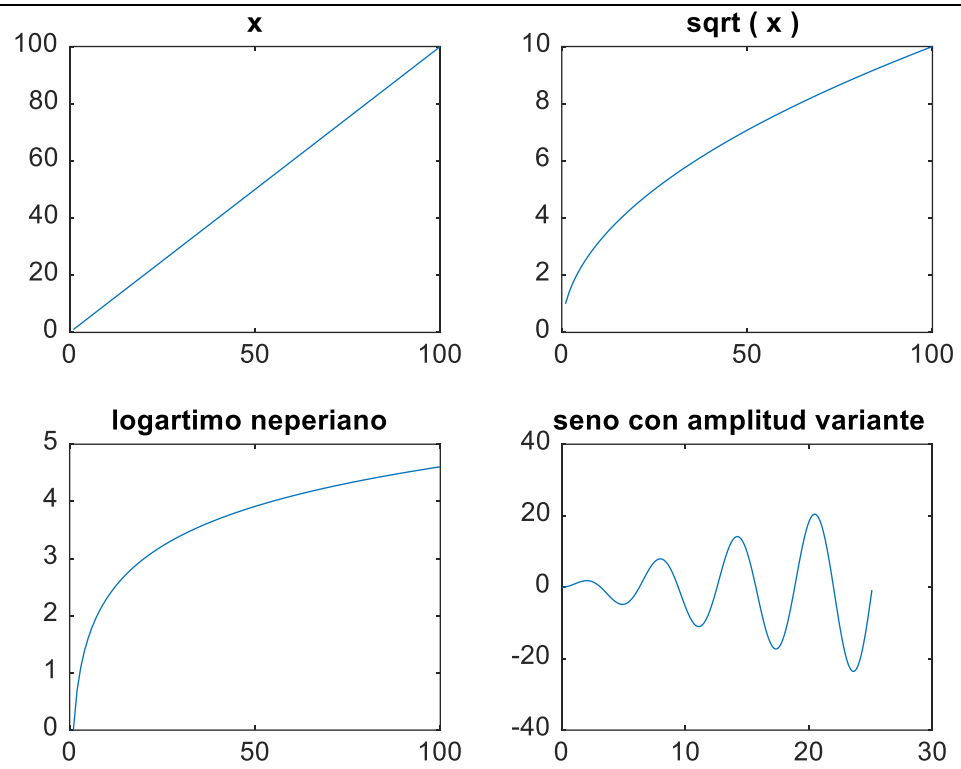

Figura 24 Ejemplo de uso de subplot.

11.9.- Gestión de varias figuras: figure

En MATLAB se utiliza el término figura (*figure*) para designar a un lugar en el que se puede dibujar. La función *plot* crea una figura nueva si no existe ninguna. En caso de que exista una, dibuja sobre ella, o la suplanta, dependiendo de cómo se haya usado *hold*. Es posible, sin embargo, trabajar con más de una figura. Para ello hay que utilizar la función *figure*, con *figure(n)* creamos una nueva figura, la Figura ***n***, o, si ya existe, la seleccionamos como la figura activa, es decir, la figura en que se dibuja. A modo de ejemplo, se crea dos figuras y dibuja un gráfico en cada uno de ellas:

```
close all % cierra todas las figuras
x=0:0.5:8*pi;
plot(x,sin(x),'o-'); % dibuja en la Figura 1
figure(2) % crea una nueva figura y la deja como activa
plot(x,sin(x)+x,'o-'); % dibuja en la figura activa (la Figura 2)
```

Como se comenta, *close all* cierra todas las figuras creadas. Otra función relacionada con la gestión de figuras es *clf*, que borra el contenido de la figura activa. Para invocar a la figura **n**, se realiza mediante el comando *figure(n)*

11.10.- Distribuciones de frecuencias: bar, pie e hist.

Existen funciones que se utilizan principalmente para generar un gráfico que permite visualizar la distribución de frecuencias de los datos de una muestra una población. *bar* y *pie* se utilizan con datos categóricos y numéricos discretos, mientras que *hist* su utiliza con valores numéricos continuos.

11.10.1.- bar

Produce un diagrama de barras, mientras que pie produce un diagrama de sectores. Supongamos que tenemos una clase con alumnos de 20 a 25 años y queremos obtener un gráfico de barras para observar visualmente la distribución de los alumnos por edad. Esto se puede hacer con las siguientes instrucciones:

```
>> edades = randi([20 25] , 1 , 20)
edades =
  Columns 1 through 17
    24    25    20    25    23    20    21    23    25    25    20    25    25    22    24    20    22
  Columns 18 through 20
    25    24    25
>> ocurrencias = histc(edades, unique(edades))
ocurrencias =
     4     1     2     2     3     8
>> bar(unique(edades), ocurrencias)
```

La primera instrucción genera un vector (fila de 20 elementos) con edades aleatorias uniformemente distribuidas entre 20 y 25 años. La segunda instrucción obtiene un vector con las ocurrencias de cada edad del vector edades, en orden creciente de edades. El resultado obtenido nos dice que en edades hay 3 elementos con el valor 20, 4 elementos con el valor 21 y así sucesivamente. La última instrucción llama a la función *bar*. El primer parámetro es un vector ordenado con las edades y el segundo es un vector con el número de ocurrencias de cada edad. El resultado de la ejecución de *bar* puede observarse en la Figura 25. Como los datos han sido generados aleatoriamente siguiendo una distribución uniforme el número de alumnos obtenidos de cada edad es parecido. La función *unique* toma como parámetro una matriz y devuelve sus elementos ordenados y sin repetición en un vector. La función *barh* produce un diagrama con barras horizontales

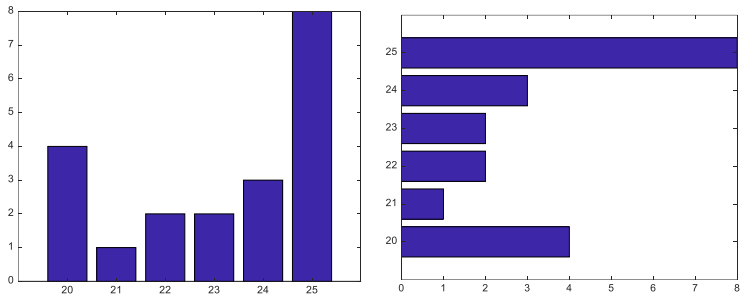

Figura 25 Resultado grafico generado por las sentencias bar(unique(edades), ocurrencias) y barh(unique(edades), ocurrencias)

11.10.2.- pie

Genera un diagrama de sectores. Por ejemplo,

```
>> subplot(1,2,1)
>> pie(ocurrencias)
>> title('Con porcentajes')
>> subplot (1,2,2)
>> pie (ocurrencias,{'20','21','22','23','24','25'})
>> title('Con etiquetas')
```

genera la Figura 26.

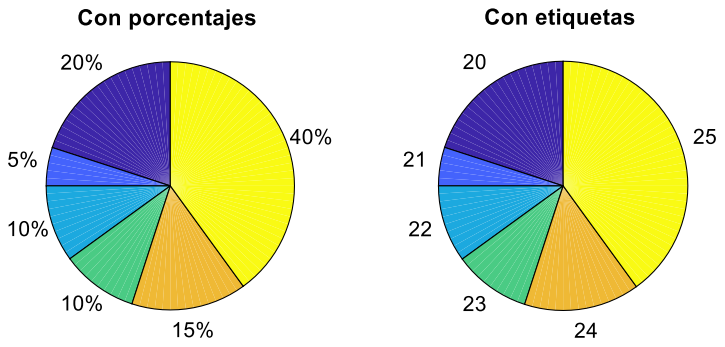

Figura 26 Diagrama de sectores con porcentajes y con etiquetas.

En la primera llamada a pie se indican las ocurrencias de cada dato y éstas se etiquetan con su porcentaje de ocurrencia. En la segunda llamada se incluye un segundo parámetro con las etiquetas de las ocurrencias. Las etiquetas se especifican como un array de celdas[7].

11.10.3.- hist

Cuando se quiere visualizar la forma de la distribución de unos datos que se representan mediante valores numéricos continuos, como la altura de un grupo de personas, entonces no se puede utilizar un diagrama de barras, puesto que la frecuencia de ocurrencia de cada dato sería muy pequeña. En ese caso se utiliza un histograma, que, a partir de los datos, crea una serie de franjas y dibuja un diagrama de barras en el que la altura o el área de cada barra es proporcional a la cantidad de datos que hay en la franja asociada a la barra. En MatLab, *hist(vector)* crea un histograma con 10 franjas de la misma anchura e *hist(vector,n)* crea un histograma con **n** franjas. Por ejemplo:

```
>> alturas = 1.74 + randn(1,100)*0.05;
>> hist(alturas)
>> ocu = hist(alturas)
ocu =

    1    3    5   22   13   22   16   11    5    2
>>
```

La primera instrucción crea un vector con 100 números aleatorios siguiendo una distribución normal de media 1.74 y desviación típica 0.05—simula 100 alturas distribuidas normalmente con media 1.74 metros y desviación típica de 5 centímetros. La segunda instrucción genera un histograma de los datos con 10 franjas, véase la Figura 27. La última instrucción muestra un segundo uso de la función *hist* en la que se obtiene un vector indicando el número de datos que caen en cada franja

[7] Es un tipo de estructuras de datos. A diferencia de los arrays numéricos, estas estructuras de datos no están especializadas en la realización de cálculos aritméticos, sirviendo para representar información genérica como los detalles de la realización de un experimento.

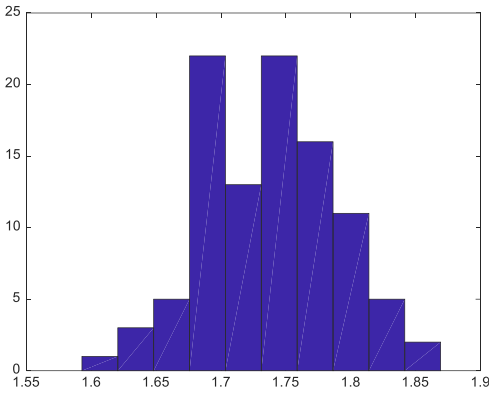

Figura 27 Histograma de alturas.

Con la sentencia

```
>> [ocu,centros] = hist( alturas)
ocu =
    1    3    5   22   13   22   16   11    5    2
centros =
   1.6066   1.6343   1.6619   1.6896   1.7172   1.7449   1.7725   1.8002   1.8278   1.8555
>>
```

obtienen los centros de las franjas.

11.11.- Función Por Partes o A Tramos

En numerosas ocasiones nos vemos en la necesidad de recurrir a MatLab® para resolver alguna situación de cálculo, ya sea en una materia como métodos numéricos, procesamiento de señales, etc…y algunas de esas situaciones involucran por lo general las conocidas funciones por partes o a tramos, cómo en la Figura 28.

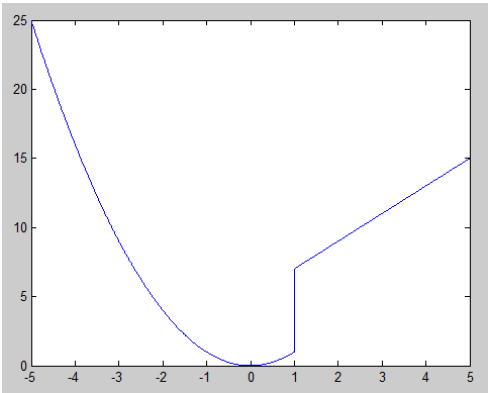

Figura 28 Gráfico de función por partes

Donde la gráfica presente en la figura, viene descrita por la siguiente función matemática.

$$f(t) = \begin{cases} t^2 & t \leq 1 \\ 2t + 5 & t > 1 \end{cases}$$

Realizar este tipo de gráfica en MatLab® es tan sencillo como se describe a continuación, inicialmente lo que necesitamos es definir el vector de la variable independiente (como es usual a la hora de graficar) lo que equivale

a definir los valores o el rango de t, es decir, ya sea utilizando la función *linspace* o cualquier otro método para generar una vector que representa los valores de la variable independiente (Figura 29) , ejemplo:

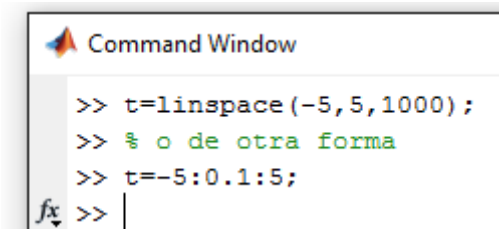

Figura 29. Definición del vector t

Para lograr el resultado de la función de la Figura 28, escribiremos el código (Figura 30) y lo explicaremos:

- Inicialmente declaramos el vector de la variable independiente.
- Posteriormente asignamos a la variable (vector) dependiente los valores respectivos, (aplicando el concepto de los operadores de igualdad y lógicos) al vector de la variable independiente, en este caso ***t***.

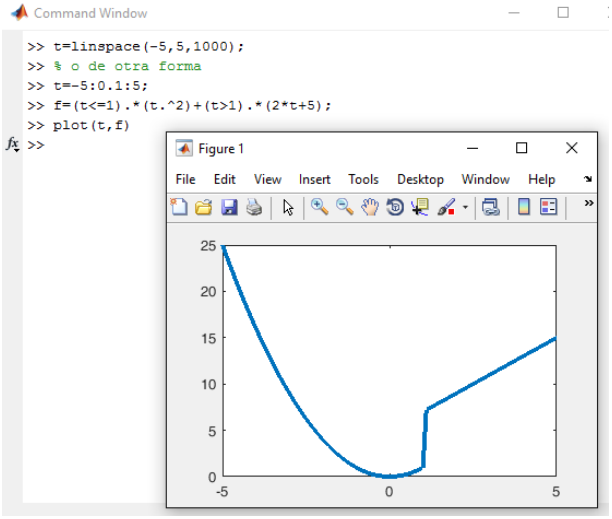

Figura 30. Función por partes y representación grafica

La primera parte de la asignación se lee (**para los *t* menores o iguales a uno**) este operación lógica nos arrojará un **true** o **"1" lógico** para los valores de ***t*** que cumplan esa condición específica y ese 1 se multiplicará por el valor correspondiente, que en este caso equivale a ***t^2***, de lo contrario, para los valores que no son menores o iguales a uno, esa operación lógica nos retornará **false** o **"0**" (cero) y anulará esa operación, pasando a la siguiente condición t>1 como ya sabemos, está condición se cumple y nos arroja un 1 lógico y el valor que se calcule en ese instante se guardará en el vector ***f***.

11.12.- Función Por Tramos Con Condición Múltiple.

A modo de ejemplo veremos una función matemática con tres condiciones

$$f(t)=\begin{cases} t+7 & t<-2 \\ -t+7 & t>2 \\ t^2 & -2\le t\le 2 \end{cases}$$

La Figura 31 muestra la implementación y su resultado gráfico. En el último término de la definición de *f*, encontramos el operador &, este es el operador lógico *and* preguntamos ¿**t mayor o igual** a -2 **y t menor o igual** 2? si ambas condiciones se cumplen (arrojan un *uno* lógico) entonces la operación completa arrojará un **1** y validará la operación ***t^2***

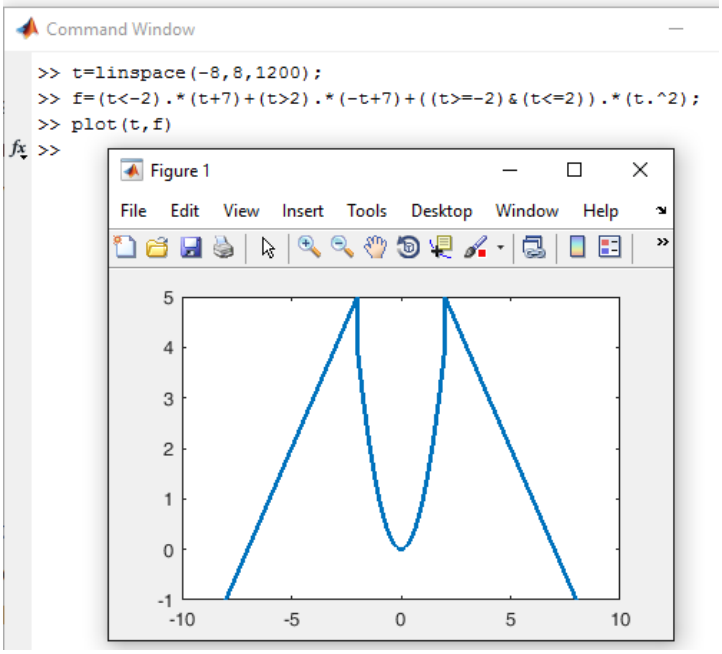

Figura 31. Función por partes con condición múltiple y su representación grafica

Bibliografía.

Help MatLab
Toolbox MatLab: http://www.mathworks.com/

Contenido