

Practica I - Parte 2

Introducción a Matlab

Introducción a las Ciencias de la Tierra y el Espacio I - 2011

Índice

1. Introducción	1
2. Comandos Básicos	2
3. Operaciones con Vectores y Matrices	2
3.1. Formas de Definir Vectores	5
3.2. Suma de matrices	5
3.3. Producto de Matrices y Producto Elemento a Elemento	5
3.4. Potencia Enésima de una Matriz	7
4. Lectura y Almacenamiento de Datos	8
4.1. Almacenamiento de Datos (<i>save</i>)	8
4.1.1. Almacenamiento en formato de texto ASCII	8
4.1.2. Almacenamiento en formato binario de Matlab	8
4.2. Creación manual de un archivo de datos	8
4.3. Lectura de Archivos de Datos (<i>load</i>)	8
4.3.1. Archivos ASCII	8
4.3.2. Archivos binarios de Matlab	9
5. Gráficas	9
5.1. Comandos para Elaborar Gráficos (<i>plot, errorbar</i>)	9
6. Creación de M-Files	11
6.1. Script-Files	11
7. Funciones Especiales	11
7.1. Medición de Coordenadas en Imágenes (<i>imread, ginput</i>)	11
7.2. Ajuste por Mínimos Cuadrados (<i>polyfit, corrcoef</i>)	14
7.3. Grabación de Audio (<i>audiorecorder, recordblocking, getaudiodata</i>)	14
7.4. Área de Poliedros (<i>ginput, polyarea</i>)	15
7.5. Distribuciones Cumulativas (<i>sort</i>)	15
8. Ejercicios	15

1. Introducción

El programa Matlab (el nombre corresponde a la abreviatura *Matrix Laboratory*) es una potente herramienta de cálculo numérico y visualización gráfica de uso muy difundido entre los científicos para el desarrollo de su tarea de investigación. Tiene la gran ventaja de ser un lenguaje de alto nivel que

Comando	Descripción
help	Lista la serie de funciones o aplicaciones que abarca Matlab (*)
pwd	Indica el directorio en el cual se esta trabajando (*)
demo	Permite acceder a las demostraciones de algunas funciones de Matlab (*)
what	Lista los M-Files existentes en el disco (*)
who	Lista las variables que se encuentran en la memoria (*)
whos	Lista las variables anteriores y sus características (*)
clc	Limpia el texto de la Ventana de Comandos
clear	Limpia todas las variables de la memoria

Cuadro 1: Comandos básicos de ayuda y navegación en Matlab. Los comandos marcados con (*) tiene sentido utilizarlos solamente en la Ventana de Comandos, no en los *M-Files*.

integra, en un único ambiente software, rutinas de cálculo, visualización y programación. El programa es de fácil uso ya que los problemas se pueden formular usando una notación matemática standard. La representación básica de los datos en Matlab es en forma matricial. 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 para su uso en aplicaciones particulares. Existen también extensiones de Matlab denominadas *Toolboxes*, que son librerías de funciones Matlab que permiten resolver problemas específicos en diversas áreas de ciencia e ingeniería. Actualmente existen *Toolboxes* en áreas tales como Control, Procesamiento de Señales, Identificación, Procesamiento de Imágenes, Redes Neuronales, Wavelets, etc. 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. Al iniciar el programa *Matlab* se desplegará una ventana desde donde se ejecutan los diferentes comandos, ver figura (1.1 superior). Para familiarizarse mas con las tareas que Matlab es capaz de hacer, podemos navegar entre los distintos temas de ayuda en el menú Help - Product Help, ver figura (1.1 inferior).

2. Comandos Básicos

El símbolo `>>` (llamado terminal, consola, prompt o línea de comandos) se muestra en la *Ventana de Comandos* (Command Window) y es el lugar donde el usuario ingresa a Matlab los comandos para que sean ejecutados. Antes de comenzar a trabajar es conveniente cambiarse al directorio de trabajo (ver figura). Siempre que se quiera acceder a mas información acerca de las tareas de Matlab y *toolboxes*, podemos A continuación veremos algunos comandos básicos de Matlab.

3. Operaciones con Vectores y Matrices

Veamos ahora como 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, ver figura (3.1 superior-izquierda). El comando `size` indica las dimensiones de la matriz (el primer número corresponde

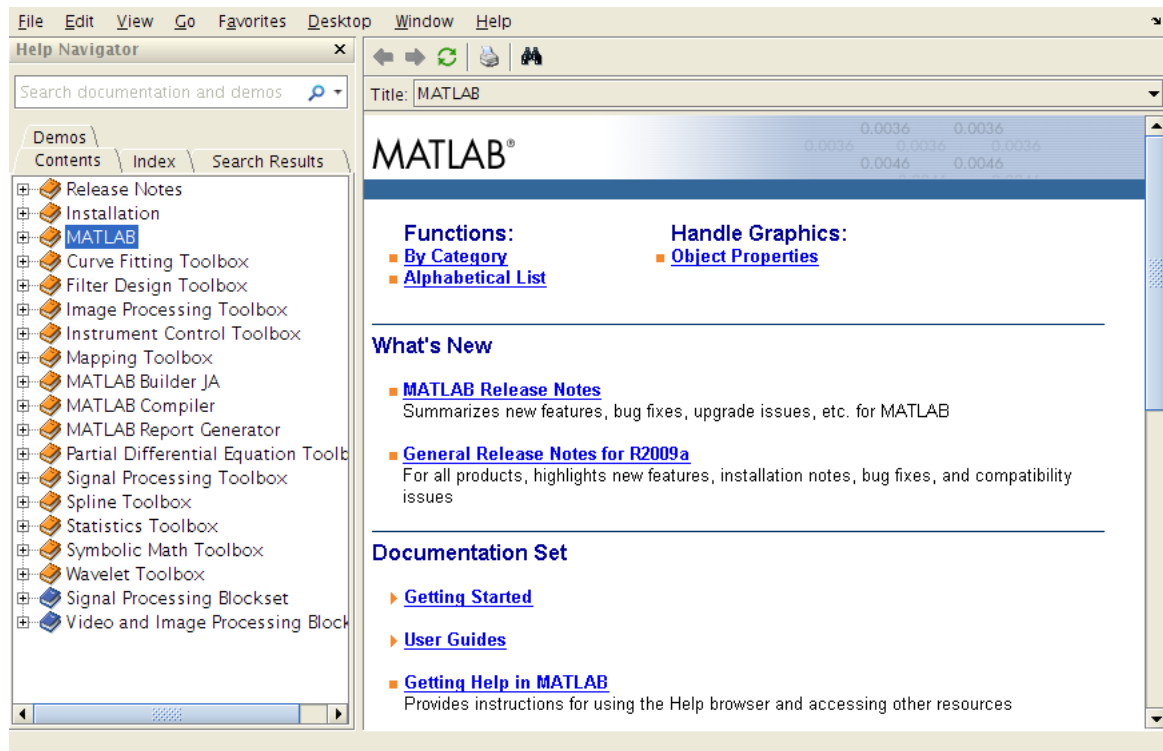
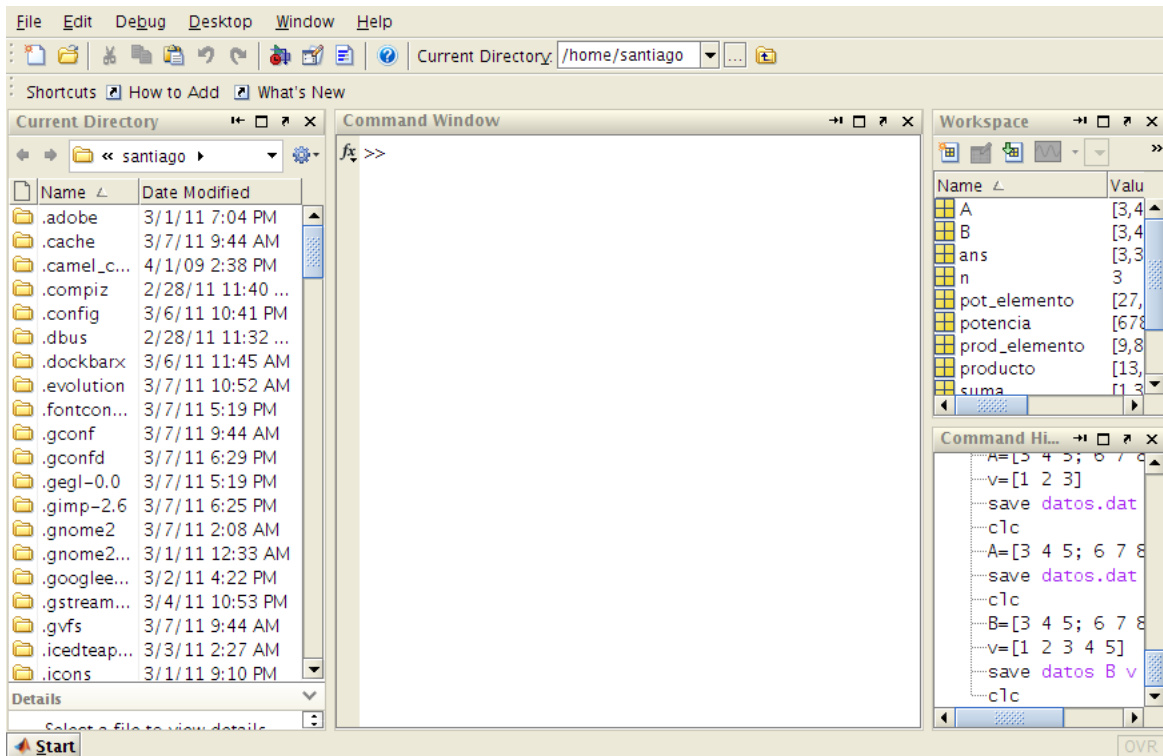


Figura 1.1: SUPERIOR: Escritorio de trabajo típico de Matlab. INFERIOR: Navegador de ayuda de Matlab con explicaciones, vínculos y ejemplos.

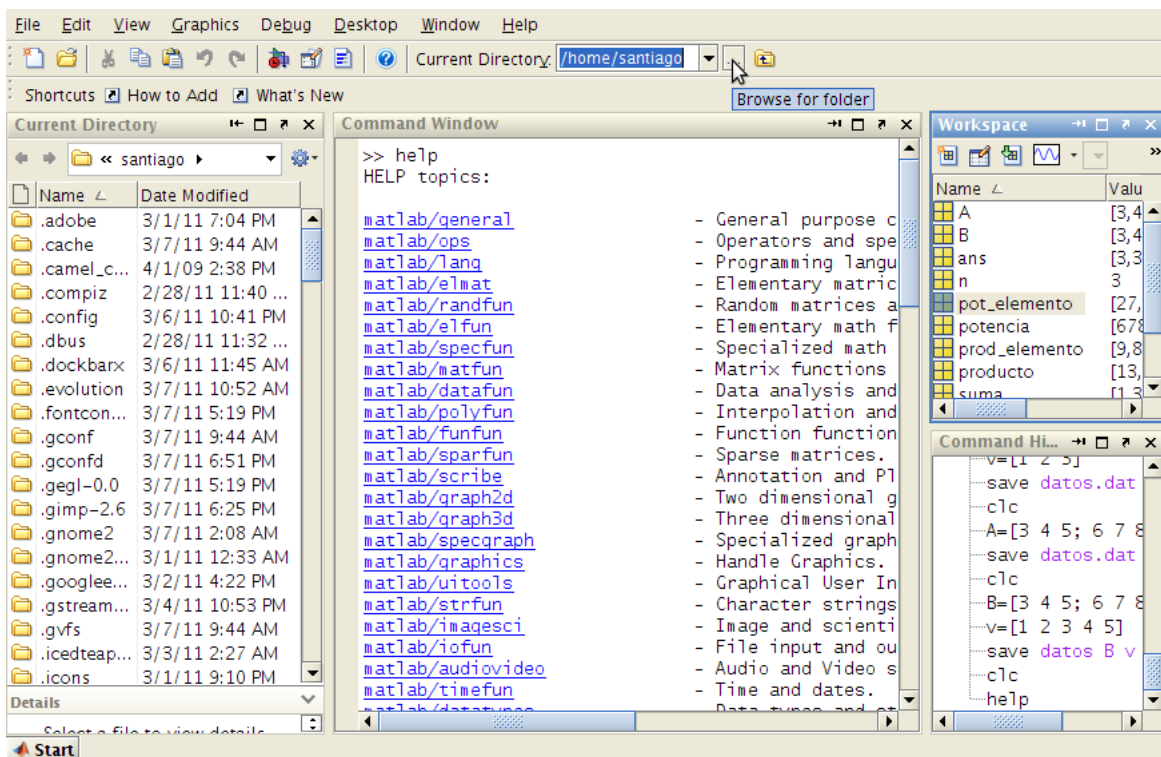


Figura 2.1: Salida del comando `help` en la Ventana de Comandos donde es posible seguir el vinculo para más ayuda acerca de la tarea deseada. Observe también en la parte superior el acceso para cambiar de directorio de trabajo del usuario o (Current Directory), señalado con el mouse.

Command Window	Command Window	Command Window
<pre>>> A=[1 3 4; 5 0 -1; 3 0 0] A = 1 3 4 5 0 -1 3 0 0 fx >> </pre>	<pre>>> size(A) ans = 3 3 fx >></pre>	<pre>>> p=A(1,3) p = 4 fx >></pre>
Command Window	Command Window	Command Window
<pre>>> v=[2 3 4 5] v = 2 3 4 5 fx >></pre>	<pre>>> v=[1:6] v = 1 2 3 4 5 6 fx >></pre>	<pre>>> v=[1:2:6] v = 1 3 5 fx >> </pre>

Figura 3.1: SUPERIOR-IZQUIERDA: En el siguiente ejemplo definimos una matriz A de 3 filas y 3 columnas (3×3), escribiendo los elementos que constituyen sus filas. SUPERIOR-CENTRO: El comando *size* indica las dimensiones de la matriz A. Para vectores el comando adecuado es *length*. SUPERIOR-DERECHA: Se muestra la forma de llamar un elemento dado de la matriz A. INFERIOR-IZQUIERDA: Se muestra la forma en que se definen los vectores. INFERIOR-CENTRO: Se muestra la forma de definir un vector indicando el comienzo y el final. INFERIOR-DERECHA: Un comando similar permite generar un vector indicando el comienzo, final y el paso de incremento entre sus elementos de esta forma se genera un vector de 1 a 6 con incremento 2.

a la cantidad de filas y el segundo a la cantidad de columnas), ver figura (3.1 superior-derecha). Si queremos identificar un elemento determinado de una matriz solo tenemos que indicar su fila y columna como se muestra en la figura (3.1 superior-derecha).

3.1. Formas de Definir Vectores

Los vectores se definen como cualquier *n-upla* de elementos en Matlab. La cantidad de elementos determina su dimensión y se definen explícitamente como se muestra en la figura (3.1 inferior-izquierda), indicando el elemento inicial, figura (3.1 inferior-centro) o de forma incremental, figura (3.1 inferior-derecha). Los vectores también pueden ser definidos a partir de las filas o columnas de una matriz. Simplemente se indica la fila o columna a llamar, utilizando el símbolo : (dos puntos) para indicar el índice recorrido y un número para indicar el índice fijo, ver figura (3.2 izquierda).

3.2. Suma de matrices

Tanto la suma como la resta solamente puede aplicarse a dos o más matrices de iguales dimensiones y la misma se realiza elemento a elemento como lo muestra la figura (3.2 centro).

3.3. Producto de Matrices y Producto Elemento a Elemento

Para efectuar el producto de las matrices A y B se debe cumplir que el número de columnas de A debe ser igual al número de filas de B como se muestra en la figura (3.2 derecha). Tal condición para efectuar el producto de matrices es también satisfecha por matrices cuadradas (3×3). En ese caso particular también es posible definir otro producto conocido como *producto elemento a elemento*. El mismo da un resultado diferente al producto usual de matrices y tiene aplicaciones varias en la manipulación de datos. Como el nombre lo dice, el producto es elemento a elemento de forma que cada

Command Window	Command Window	Command Window
<pre>>> A A = 1 3 4 5 0 -1 3 0 0 >> u=A(2,:) u = 5 0 -1 >> w=A(:,1) w = 1 5 3 fx >></pre>	<pre>>> A=[1 3 4; 5 0 -1; 3 0 0]; >> B=[0 0 5; -5 -3 1/2; 1 -1 2]; >> suma=A+B suma = 1.0000 3.0000 9.0000 0 -3.0000 -0.5000 4.0000 -1.0000 2.0000 fx >> </pre>	<pre>>> A=[3 4 5; 6 7 8] A = 3 4 5 6 7 8 >> B=[3 2; 1 1; 0 -3] B = 3 2 1 1 0 -3 >> producto=A*B producto = 13 -5 25 -5 fx >> </pre>

Figura 3.2: IZQUIERDA: Diferentes formas de definir vectores a partir de una matriz cualquiera. CENTRO: Operación de suma de matrices, observe que dichas matrices deben ser de iguales dimensiones. DERECHA: Operación de producto de matrices. Observe que el número de columnas de A es igual al número de filas de B.

Command Window	Command Window	Command Window
<pre>>> A=[3 4 5; 6 7 8; 3 1 0] A = 3 4 5 6 7 8 3 1 0 >> B=[3 2 1; 1 0 -3; 3 2 1] B = 3 2 1 1 0 -3 3 2 1 >> prod_elemento=A.*B prod_elemento = 9 8 5 6 0 -24 9 2 0 fx >></pre>	<pre>>> A=[3 4 5; 6 7 8; 3 1 0] A = 3 4 5 6 7 8 3 1 0 >> n=4 n = 4 >> potencia=A^n potencia = 6789 6698 7207 12126 11975 12892 2661 2651 2868 fx >> </pre>	<pre>>> B=[3 2 1; 1 0 -3] B = 3 2 1 1 0 -3 >> n=3 n = 3 >> pot_elemento=B.^n pot_elemento = 27 8 1 1 0 -27 fx >></pre>

Figura 3.3: IZQUIERDA: Operación de producto de matrices elemento a elemento. CENTRO: Potencia de una matriz. Observe que equivale al producto usual de la matriz con ella misma, por lo tanto tiene que ser cuadrada. DERECHA: Potenciación elemento a elemento de una matriz.

The image shows three windows from a MATLAB environment. The top-left window shows the creation of a 3x5 matrix A and its saving to 'datos.dat' in ASCII format. The bottom-left window shows the creation of matrix B and vector v, and their saving to 'datos' in binary format. The center window shows the contents of 'datos.txt', which is a 7x2 table of data. The right window shows the loading of 'datos.txt' into MATLAB, resulting in a column vector x and a row vector y.

```

Command Window
>> A=[3 4 5; 6 7 8; 3 1 0]

A =

     3     4     5
     6     7     8
     3     1     0

>> save datos.dat A -ascii
fx >>

Command Window
>> B=[3 4 5; 6 7 8; 3 1 0]

B =

     3     4     5
     6     7     8
     3     1     0

>> v=[1 2 3 4 5]

v =

     1     2     3     4     5

>> save datos B v
fx >>

datos.txt
1      0.5
2      0.7
3      1.2
4      1.5
5      1.7
6      2.1
7      3.4

Command Window
>> load datos.txt
>> x=datos(:,1)

x =

     1
     2
     3
     4
     5
     6
     7

>> y=datos(:,2)

y =

    0.5000
    0.7000
    1.2000
    1.5000
    1.7000
    2.1000
    3.4000

fx >>

```

Figura 3.4: SUPERIOR-IZQUIERDA: Sintaxis para guardar una sola variable a la vez en formato de texto ASCII. INFERIOR-IZQUIERDA: Sintaxis para guardar una o más variables en formato binario de Matlab. CENTRO: Vista de un procesador de texto plano (como el Block de Notas) donde puede generarse un archivo de datos en formato ASCII. DERECHA: Comando para cargar los datos en Matlab y asignar como vectores las distintas columnas de la matriz de datos.

cada elemento está definido por el producto de los elementos correspondientes en las matrices dadas. Observe que entre el símbolo de producto \star (asterisco), se antepone un \bullet (punto) para diferenciarlo del producto usual de matrices.

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

3.4. Potencia Enésima de una Matriz

Algo similar a lo expuesto para el producto de matrices. La potenciación usual se denota con el símbolo \wedge (casita) como se muestra en la figura (3.3 centro). La matriz a elevar tiene que ser cuadrada. Análogamente la potenciación elemento a elemento simplemente opera elevando a la potencia cada elemento por separado y es aplicable a cualquier tipo de matrices. Nuevamente para indicar que la operación es elemento a elemento se antepone un \bullet (punto) antes de la \wedge (casita), ver figura (3.3 derecha).

4. Lectura y Almacenamiento de Datos

4.1. Almacenamiento de Datos (*save*)

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* pero difiere la sintaxis de acuerdo la forma que queramos guardarlo, formato ASCII, ver figura (3.4 superior-izquierda), o formato binario de Matlab, ver figura (3.4 inferior-izquierda).

4.1.1. Almacenamiento en formato de texto ASCII

La sintaxis es

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

La ventaja de usar este método radica en que este archivo de datos puede ser leído por cualquier programa de manejo de texto y/o planillas de cálculo. Por ejemplo: Block de Notas, Excel, etc. La terminación (.dat) no es obligatoria (también se suele usar la terminación .txt), pero se suele utilizar (.dat) 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.

4.1.2. Almacenamiento en formato binario de Matlab

La sintaxis es

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

Por defecto, Matlab coloca a estos archivos la terminación (.mat) para indicar que el formato es de Matlab. Estos archivos no pueden ser leídos desde programas de procesamiento de texto, pero tienen algunas ventajas que se verán más adelante. En este caso no es necesario que todas las variables tengan la misma dimensión para ser almacenadas.

4.2. Creación manual de un archivo de datos

Suele ocurrir que los datos experimentales que se obtienen en el laboratorio son anotados en libretas o cuadernos que suelen ser pasados al PC. Una forma cómoda para poder utilizar estos datos posteriormente con Matlab es generar un archivo de datos desde un editor de texto elemental o texto plano (o en inglés *plain text*) como el Block de Notas, ver figura (3.4 centro). Los archivos generados como texto plano son de formato ASCII. Para generarlo simplemente se debe abrir el Block de Notas e ingresar los datos en forma de columnas separadas por espacios o tabulaciones. Al almacenar tener la precaución de ponerle terminación (.dat) o (.txt) para identificarlos rápidamente como archivo de datos.

4.3. Lectura de Archivos de Datos (*load*)

El comando para leer archivos de datos es *load* y también difiere en la sintaxis según se trate de un archivo de datos en formato ASCII, ver figura (3.4 derecha) o binario de Matlab.

4.3.1. Archivos ASCII

La sintaxis es

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

Notemos que si hacemos un *whos*, la variable que tenemos en la memoria del Matlab tiene el mismo nombre que el archivo, pero sin la terminación.

COMANDO	DESCRIPCIÓN
<code>figure</code>	Genera una nueva pantalla de gráficos
<code>plot</code>	Crea un gráfico simple
<code>plotyy</code>	Permite plotear dos set de datos, ambos con eje de las Y a la izquierda y derecha
<code>loglog</code>	Permite graficar en escala logarítmica en ambos ejes
<code>semilogx</code>	Permite graficar en escala logarítmica en el eje de las X y lineal en el eje de las Y
<code>semilogy</code>	Permite graficar en escala lineal en el eje de las X y logarítmica en el eje de las Y
<code>subplot</code>	Permite dividir la ventana gráfica en cuadrantes y hacer varias gráficas en él
<code>ezplot</code>	Permite crear un gráfico de variable simbólica
<code>errorbar</code>	Crea un gráfico con barras de error
<code>hold</code>	Permite realizar la superposición de dos o más gráficos en una misma pantalla
<code>grid</code>	Activa o desactiva la grilla automática en la gráfica
<code>title</code>	Ingresa un título a la gráfica
<code>xlabel</code>	Ingresa un la etiqueta del eje de las X
<code>ylabel</code>	Ingresa un la etiqueta del eje de las Y
<code>text</code>	Permite ingresar texto en la gráfica en coordenadas definidas por el usuario
<code>axis</code>	Permite ajustar el rango de valores en X e Y y también su proporcionalidad
<code>legend</code>	Ingresa una leyenda para los datos experimentales

Cuadro 2: Lista de comandos gráficos en Matlab. Para aprender como es la sintaxis del comando se debe ingresar `help` seguido del comando en la Ventana de Comandos.

4.3.2. Archivos binarios de Matlab

La sintaxis es

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

Otra opción para las versiones más nuevas es

```
a = load('nombre de archivo')
```

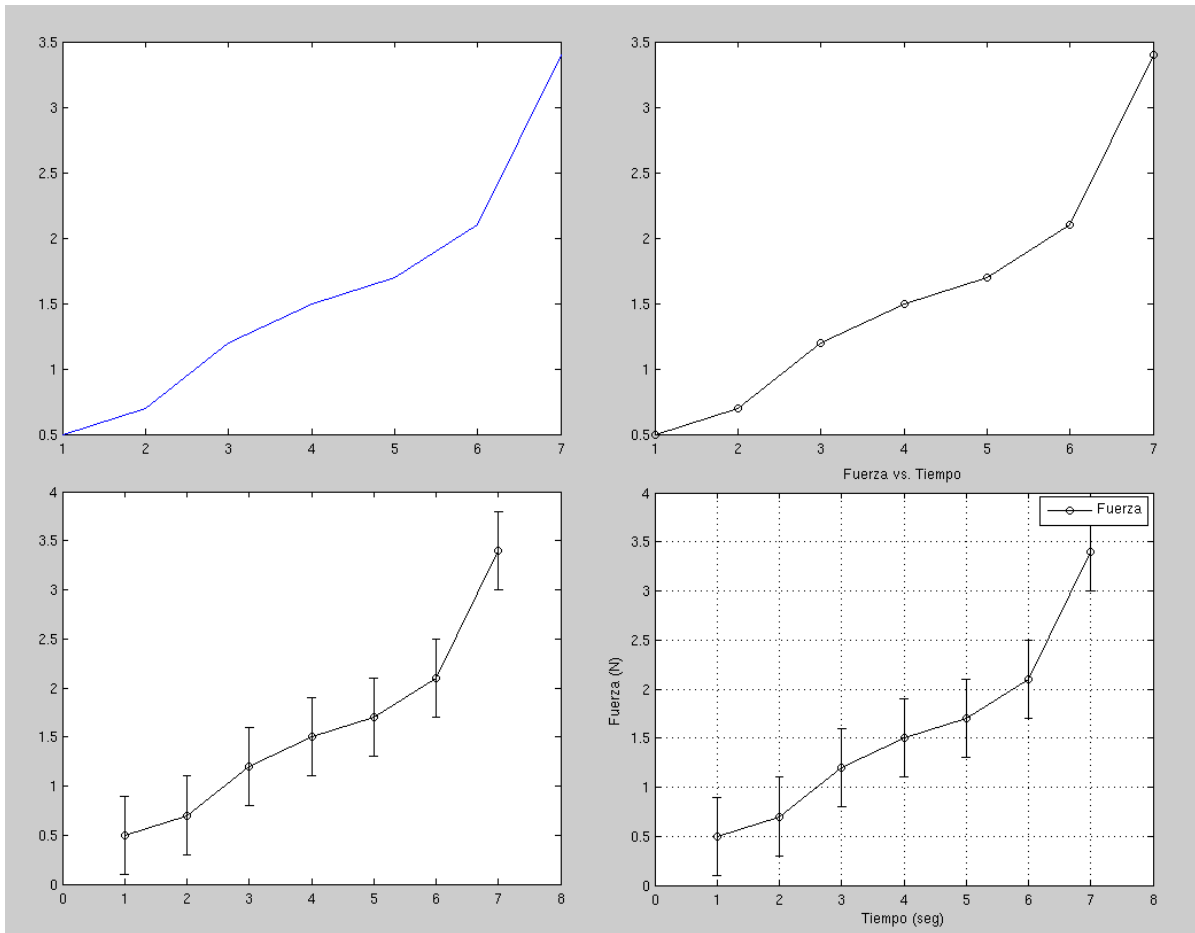
De esta forma la variable contiene los datos almacenados en el archivo. Notemos que si hacemos un `whos`, las variables que tenemos en la memoria del Matlab tiene el mismo nombre con que habían sido almacenadas, lo que representa una gran ventaja si se está realizando la lectura en el marco de un programa, puesto que se conoce a priori el nombre de las variables.

5. Gráficas

Matlab tiene un excelente manejo de gráficos. Aquí veremos sólo algunos de los comandos básicos para la generación de gráficas en dos dimensiones. Para aquellos estudiantes que tengan interés, recomendamos ejecutar `help graphics`, `help graph2d` y `help graph3d`.

5.1. Comandos para Elaborar Gráficos (`plot`, `errorbar`)

En el cuadro (2) se listan una serie de comandos gráficos. Algunos pueden ser utilizados en secuencia para graficar una serie de datos y obtener una gráfica final con todas las leyendas y etiquetas correspondiente, como se muestra en la tabla de comandos de la figura (5.1). Observe que para ejecutar el comando `errorbar`, es necesario un tercer vector e , de igual longitud que x e y , que contiene el error de y .



PLOTEIO	COMANDOS INGRESADOS
SUPERIOR-IZQUIERDA	<code>plot(x,y)</code>
SUPERIOR-DERECHA	<code>plot(x,y,'ok-')</code>
INFERIOR-IZQUIERDA	<code>errorbar(x,y,e,'ok-')</code>
INFERIOR-DERECHA	<code>errorbar(x,y,e,'ok-')</code> <code>title('Fuerza vs. Tiempo')</code> <code>xlabel('tiempo (seg)')</code> <code>ylabel('Fuerza (N)')</code> <code>legend('Fuerza')</code> <code>grid on</code>

Figura 5.1: SUPERIOR-IZQUIERDA: Ploteo simple de dos vectores. SUPERIOR-DERECHA: Ploteo con opción de color, punto y línea. INFERIOR-IZQUIERDA: Ploteo con opciones de color, punto, línea, y barra de error. INFERIOR-DERECHA: Ploteo con opciones de color, punto, línea, barra de error, grilla, título, leyenda y nombre en los ejes X e Y.

6. Creación de M-Files

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*. La forma de editar *M-files* es usando un editor incorporado a Matlab (versión 5 o posteriores), el denominado *Matlab Editor/Debugger*, al cual se accede desde el menú de archivo.

6.1. Script-Files

Un *script-file* consiste de una sucesión de líneas de comando tal como las ingresaríamos en la Ventana de Comandos de Matlab para su ejecución en tiempo real. Por ejemplo, si el archivo tiene el nombre `temp_conv.m`, como se muestra en la figura (6.1 superior), el mismo puede ser ejecutado desde la Ventana de Comandos y todas las líneas que del programa se ejecutarán en el orden en que aparecen. Las variables definidas en el script-file son globales por lo tanto cambiarán el valor de aquellas variables que estén definidas con el mismo nombre antes de ser ejecutado el programa. Para evitar confusiones se suele encabezar el programa con las tres líneas que aparecen en la figura (6.1 superior). Las mismas tienen como tarea: limpiar la pantalla, cerrar todas las ventanas gráficas y limpiar las variables de la memoria respectivamente. Los script-files también pueden ser creados en editores de texto convencionales de los que se mencionaron anteriormente en esta práctica. Para poder ver que es lo que el programa está haciendo en medio de la ejecución, los script-files pueden imprimir texto en la ventana de comandos con el comando *disp*, como ser mensajes de bienvenida, aviso de algún error, etc. Cuando el programa es muy extenso es muy usual hacer anotaciones dentro del mismo para guiarnos durante la programación. Tales comentarios o anotaciones no deben ser interpretadas por Matlab a la hora de ser ejecutado el programa. Esto se logra anteponiendo el símbolo % (por ciento) delante del comentario que automáticamente se pintará de color verde. En el cuadro (3) se muestran una serie de comandos típicos de programación de script-files a los cuales se le suma todos los comandos gráficos, los comandos para guardar y cargar datos en archivos (.dat) o (.txt) anteriormente citados.

7. Funciones Especiales

Existen infinidad de funciones y tareas especializadas de Matlab que permiten el análisis de imágenes, sonido y otras formas en las que pueden presentarse los datos científicos. Dado que este manual no pretende abarcar todas las funciones (existen manuales/libros más completos) solamente citaremos algunas funciones especiales que serán utilizadas en las prácticas a lo largo del curso.

7.1. Medición de Coordenadas en Imágenes (*imread*, *ginput*)

Para muchas tareas es muy útil poder medir pares de coordenadas (x, y) de una imagen determinada. Para hacer esto en Matlab primero se debe setear el directorio de trabajo en la misma carpeta donde se encuentra la imagen a medir. Luego abrir la imagen como una figura en Matlab con el siguiente comando. Preferentemente trabajar con imágenes en formato *Joint Photographics Experts Group* (JPEG) cuya terminación suele ser `.jpg`

```
A=imread('imagen.jpg','jpg')
```

En este momento se desplegará la imagen en una ventana de una figura. Observe que la relación de aspecto entre los ejes no es la misma, esto ocasiona que la imagen se vea ensanchada o alargada. Para evitar esto y que la imagen se visualice correctamente hay que ejecutar el siguiente comando

```
axis equal
```

```

1 - clc
2 - close all
3 - clear all
4
5 - disp('Bienvenido al conversor de grados Fahrenheit a Celsius')
6 - disp('-----')
7
8 % Esto es un comentario...
9
10 - f=input('Ingrese la temperatura en grados Celsius: ');
11 - c=F*(9/5)+32;
12
13 % c es la temperatura en celsius
14
15 - disp('La temperatura en Fahrenheit es')
16 - c

```

Current Directory: /home/santiago/Matlab

Command Window

```

Bienvenido al conversor de grados Fahrenheit a Celsius
-----
Ingrese la temperatura en grados Celsius: 36.5
La temperatura en Fahrenheit es

c =

    97.7000

fx >>

```

Name	Value
ans	97.7000
c	97.7000
f	36.5000

Command History

```

legend('Fuerza')
legend('Fuerza')
help subplot
axes
axis
help axis
clc
cd Matlab/
clc
temp_conv
36.5

```

Figura 6.1: SUPERIOR: Vista del editor *Matlab Editor/Debugger* donde se muestra un programa *M-File* de tipo *Script-File*. El programa */script* ha sido guardado bajo el nombre de *temp_conv.m* INFERIOR: La ejecución del mismo programa en la ventana de comandos. El programa ha sido ejecutado simplemente escribiendo *temp_conv* en la Ventana de Comandos.

COMANDO	DESCRIPCIÓN
<code>clc</code>	Limpia el texto de la Ventana de Comandos
<code>clear all</code>	Limpia todas las variables de la memoria
<code>close all</code>	Cierra todas las ventanas gráficas abiertas
<code>disp</code>	Escribe texto en la Ventana de Comandos durante la ejecución
<code>input</code>	Permite que el usuario ingrese un valor que es almacenado como una variable
<code>if</code>	Abre la sentencia condicional
<code>for</code>	Abre la tarea de bucle o repetición
<code>switch</code>	Similar a IF, permite ramificar la tarea según el valor de alguna variable
<code>end</code>	Finaliza sentencias IF, FOR, SWITCH
<code>menu</code>	Permite elegir al usuario que opción seguir ante una ramificación
<code>export</code>	Permite exportar datos o variables en distintos formatos ASCII
<code>zeros</code>	Crea una matriz de elementos todos 0
<code>ones</code>	Crea una matriz de elementos todos 1
<code>eye</code>	Crea una matriz identidad
<code>min</code>	Devuelve el mínimo elemento de un vector
<code>max</code>	Devuelve el máximo elemento de un vector
<code>mean</code>	Calcula el promedio de los elementos de un vector
<code>std</code>	Calcula la desviación estándar entorno al promedio de los elementos de un vector
<code>mode</code>	Calcula la moda entre los elementos de un vector
<code>sum</code>	Calcula la suma de todos los elementos de un vector
<code>length</code>	Devuelve la longitud de un vector
<code>size</code>	Devuelve las dimensiones de una matriz en filas y columnas
<code>polyfit</code>	Realiza un ajuste por un polinomio de grado n y calcula los coeficientes del polinomio
<code>corrcoef</code>	Calcula el coeficiente de correlación de <i>Pearson</i> para una serie de datos experimentales
<code>inv</code>	Calcula la inversa de una matriz cuadrada con determinante no nulo

Cuadro 3: Comandos de programación y su descripción. Para aprender como es la sintaxis del comando se debe ingresar *help* seguido del comando en la Ventana de Comandos. Este cuadro se complementa con el cuadro (2) de comandos gráficos.

Ahora la imagen esta correctamente visualizada y ya podemos comenzar con las medidas dentro de la imagen. Para realizar esto definimos 2 vectores x e y que contengan los puntos a registrar con la función *ginput*.

```
[x,y]=ginput
```

7.2. Ajuste por Mínimos Cuadrados (*polyfit*, *corrcoef*)

Una vez cargados en memoria los vectores de datos con los que se desea realizar un ajuste por mínimos cuadrados, es posible realizar el mismo de una manera muy simple. Arbitrariamente x podría ser el tiempo e y una magnitud física cualquiera, por ejemplo, una temperatura o una velocidad. Tales vectores deberán ser ingresados en Matlab y el ajuste debe hacerse de la siguiente forma

```
p=polyfit(x,y,1) % para un ajuste lineal
p=polyfit(x,y,2) % para un ajuste cuadrático
```

Esto creara una variable nueva P que es un vector que contiene 2 elementos. Los mismos son el *coeficiente angular* y la *ordenada en el origen* de la recta que mejor ajusta los datos. La sintaxis de la tarea *polyfit* involucra los vectores x e y y el grado del polinomio al cual se quieren ajustar los datos. En un caso mas general la tarea puede aplicarse para encontrar la mejor parábola que ajusta los datos, o polinomios de mayor grado, simplemente el vector P tendrá $n + 1$ elementos donde n es el grado del polinomio. En nuestro ejemplo $n = 1$. Luego para conocer el valor de las constantes ajustadas simplemente nos fijamos en los valores de los elementos del vector P . Si el ajuste es lineal, los mismos serán

```
p(1) % coeficiente angular
p(2) % ordenada en el origen
```

Para conocer el valor del *Coefficiente de Correlación de Pearson* r operamos con la tarea *corrcoef*

```
corrcoef(x,y)
```

La misma devolverá una matriz cuyos elementos de la diagonal serán 1. Los elementos en los extremos de la anti-diagonal contienen el valor de r .

7.3. Grabación de Audio (*audiorecorder*, *recordblocking*, *getaudiodata*)

Matlab es capaz de grabar sonido a través de un microfono conectado a la PC para su posterior análisis. Para efectuar la grabación previamente hay que definir el objeto a grabar y posteriormente capturarlo mediante los siguientes comandos.

```
sonido=audiorecorder(44100,16,1)
```

Donde 44100Hz es la *SampleRate*, 16 es la profundidad de *bits* del sonido y 1 es el el número de canales en los cuales se grabará el sonido (*mono* o *stereo*), en este caso *mono*.

```
recordblocking(sonido, 10);
```

Aquí especificamos que vamos a grabar el objeto *sonido*, durante 10 segundos de tiempo.

```
[y,t]=getaudiodata(sonido);
```

Luego los 2 vectores creados contienen la intensidad de sonido en y , y el instante de tiempo en el que se da la señal en el vector t .

7.4. Área de Poliedros (*ginput*, *polyarea*)

Si se quiere calcular el área de un cierto poliedro en una imagen, es posible hacerlo en Matlab. Para esto definimos 2 vectores x e y que contengan los puntos a registrar con la función *ginput*.

```
[x,y]=ginput
```

Calculamos el área con la función *polyarea* de esta forma

```
area=polyarea(x,y)
```

Para efectuar esta tarea sin problemas no pueden haber vértices con las mismas coordenadas. El hecho que hayan vértices iguales genera ambigüedades en el área del poliedro.

7.5. Distribuciones Cumulativas (*sort*)

Se pretende determinar el número de objetos mayores que un valor dado $N(>D)$. Con el comando *sort* se ordena los valores de mayor a menor.

```
D=sort(D,1,'descend');
```

8. Ejercicios

Ejercicio 1

1. Definir un vector fila de por lo menos 10 elementos
2. Definir un vector columna de por lo menos 10 elementos
3. Definir un vector columna de por lo menos 10 elementos
4. Elevar al cuadrado cada uno de los elementos del vector definido en (1)
5. Calcular el logaritmo de cada uno de los elementos del vector definido en (2)
6. Almacenar los datos en un archivo binario cuyo nombre sea *ejercicio1p1*

Ejercicio 2

1. Crear un vector v cuyo primer elemento sea 55, el último 480 y tal que la diferencia entre dos elementos consecutivos sea 5
2. Definir una variable n que contenga el número de elementos del vector v
3. Definir un vector u que contenga la raíz cúbica de los elementos de v
4. Transponer los vectores definidos en (1) y en (3)
5. Definir los siguientes vectores tales que: $q = n \star v$, $s = v \star u$ y $t = u/v$
6. Definir una variable que contenga la suma de los elementos del vector q
7. Almacenar todas las variables definidas en un archivo binario cuyo nombre sea *ejercicio2p1*

Ejercicio 3

Dadas las siguientes matrices, realizar las siguientes operaciones

1. $A + B - C$
2. $A \star B$
3. C^2
4. Elevar cada uno de los elementos de la matriz C al cubo
5. Calcular la matriz inversa de A
6. Calcular el determinante de B
7. Definir una nueva matriz D tal que $d_{ij} = a_{ij} \star b_{ij}$
8. Almacenar todas las variables en un archivo ASCII cuyo nombre sea *ejercicio3p1*

$$A = \begin{pmatrix} 3 & 0 & -2 \\ 1 & 4 & 5 \\ -1 & 1 & 2 \end{pmatrix}, B = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 6 & 1 \\ 3 & -2 & -5 \end{pmatrix}, C = \begin{pmatrix} -1 & -1 & 2 \\ 5 & 1 & 1 \\ -3 & -2 & 3 \end{pmatrix}$$

Ejercicio 4

1. Lea el archivo *matriz.dat* guardado en el disco
2. Determine las dimensiones de la matriz y defina una variable n para el número de filas y una variable m para el número de columnas
3. Seleccione los elementos de la segunda y cuarta filas, y los elementos de la segunda columna (guárdelos en forma de vector)
4. Defina una matriz de dimensión $(n \times m)$ de ceros y una matriz de igual dimensión a la anterior cuyos elementos sean todos 1
5. Almacenar las variables en un archivo cuyo nombre sea *ejercicio4p1*
6. Grafique la segunda columna en función de la cuarta y la primera columna en función de la tercera, utilizando diferentes tipo de símbolos y colores. Etiquete los ejes y asigne un título y leyenda a la gráfica
7. Almacenar la gráfica con el nombre *grafica4p1.fig*

Ejercicio 5 - Ley de Hubble (obligatorio)

Algo sumamente importante en astronomía es poder establecer relaciones de causalidad entre variables de un modelo. En 1929 *Edwin Hubble* relacionó la distancia a objetos brillantes con su velocidad radial, es decir la velocidad en la cual el objeto se acerca o aleja del observador. Teniendo en cuenta el *Efecto Doppler*, esquematizado en la figura (8.1), que establece que los objetos que se alejan de nosotros tienen un corrimiento en las líneas espectrales hacia el rojo y los que se alejan hacia el azul, *Hubble* llegó a la conclusión que todos los objetos observados se alejaban.

La importancia del planteo de *Hubble* es que apoya observacionalmente el modelo de expansión del Universo y el *Big Bang*. A través de esta actividad se podrá reproducir el trabajo de Hubble de una manera muy simplificada.

1. Cargue el vector *h1.txt* donde se encuentran los datos correspondientes a la distancia (en Mpc) y velocidad radial (km/s)

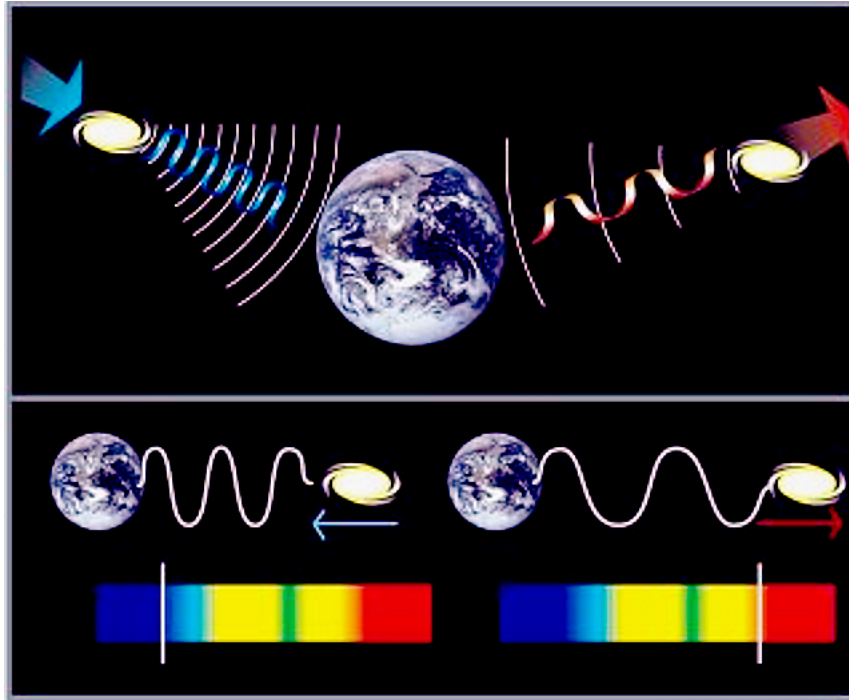


Figura 8.1: Esquema del Efecto Doppler y el corrimiento de las líneas espectales

2. Elabore una gráfica de Velocidad Radial vs. Distancia
3. Establezca una relación funcional entre las variables. Por ejemplo, si se considera una relación lineal, puede utilizar el comando `polyfit` para ajustar los datos con un polinomio de orden n
4. Superponga la gráfica de la función hallada con los datos originales. Puede usar los comandos `polyval` o `ezplot`
5. Deduzca que tan confiable es su ajuste
6. Como expresaría a través de una ecuación el resultado obtenido?
7. En base a su respuesta anterior, calcule el error correspondiente