



Curso Básico de MATLAB
Ingeniero Marcelo Bruno
Departamento de Ingeniería Eléctrica
U.N.S.

I.Instalación de Matlab

- 1.1.Instalación*
- 1.2.Procedimiento de instalación*
- 1.3.Instrucciones de instalación*
- 1.4.Inicio de Matlab*
- 1.5.Como terminar Matlab*
- 1.6.Estructura de directorios*
- 1.7.Uso del Help*

II.Descripción del Ambiente de Trabajo

- 2.1.Escritorio de Matlab*
 - 2.2.Herramientas del escritorio*
 - 2.2.1.Command Window*
 - 2.2.2.Command History*
 - 2.2.3.Launch Pad*
 - 2.2.4.Help Browser*
 - 2.2.5.Current Directory Browser*
 - 2.2.6.Workspace Browser*
 - 2.2.7.Editor/Debugger*
 - 2.3.Ejecución de Funciones*
 - 2.3.1.Ejecución de funciones e ingreso de variables*
 - 2.3.2.Ejecución de múltiples funciones*
 - 2.3.3.Ejecución de funciones largas*
 - 2.3.4.Colores de Texto*
 - 2.3.5.Como limpiar la ventana de Comandos*
 - 2.3.6.Eliminando salidas de resultado*
 - 2.3.7.Paginación de la ventana de Comandos*
 - 2.3.8.Ejecución de Programas*
 - 2.3.9.Guardar sesiones de Trabajo*
 - 2.4. Editor de arreglos y matrices*
 - 2.4.1.Diferencia entre arreglos y matrices*
 - 2.4.2.Editor*
 - 2.5. Importación y Exportación de Datos*
 - 2.5.1. Importación de datos con formato de texto*
 - 2.5.2. importación de datos binarios*
 - 2.5.3.Exportación de Datos ASCII*
 - 2.5.4.Exportación de Datos Binarios*
 - 2.6. Excel Link*
 - 2.6.1.Instalación de Excel Link*
 - 2.6.2.Funciones.*
- Ejercicios*

III. Funciones Generales y Matemáticas de Matlab

- 3.1. Variables
 - 3.2. Números y representaciones
 - 3.3. Operadores Aritméticos
 - 3.4. Operadores Matriciales vs operadores Elemento a Elemento.
 - 3.5. Generación de Vectores y Matrices
 - 3.6. Concatenación de Vectores y Matrices
 - 3.7. Operaciones con filas y columnas
 - 3.8. Funciones básicas de matrices y vectores
 - 3.9. Funciones de formato de salida
 - 3.10. Operaciones con cadenas de caracteres
- Ejercicios

IV. Lenguaje de Programación de Matlab

- 4.1. Operadores de Relación
- 4.2. Operadores Lógicos
- 4.3. Regla de precedencia de operadores
- 4.4. Sentencias de Control de Flujo
- 4.5. Programación en Matlab
 - 4.5.1. Que es un archivo M
 - 4.5.2. Script
 - 4.5.3. Función
- 4.6. Variables Locales y Globales
- 4.7. Tipos de Datos en Matlab
- 4.8. Editor/Depurador (Editor/Debugger)

V. Realización de Graficos en Matlab

- 5.1. Como graficar una función
 - 5.1.1. Estilo de líneas y marcadores
 - 5.1.2. Agregar una grafica a una grafica existente
 - 5.1.3. Graficar solamente los puntos de datos
 - 5.1.4. Grafica de datos de matrices
 - 5.1.5. Grafica de datos imaginarios
 - 5.1.6. Grafica con dos ejes Y
- 5.2. Presentación de la grafica
(Adecuación de los ejes y presentación de escalas)
- 5.3. Ventanas de figuras (subgraficas)
- 5.4. Formato de la grafica
 - 5.4.1. Titulo de la Grafica
 - 5.4.2. Legenda
 - 5.4.3. Etiquetas de los ejes
 - 5.4.4. Anotaciones de Texto
 - 5.4.5. Flechas y líneas
 - 5.4.6. Estadística de la Grafica
- 5.5. Impresión y Exportación

1.1 Instalación

Antes de la instalación

Antes de ejecutar el programa instalador se deben tomar los siguientes recaudos:

- . Desactivar cualquier software antivirus que este instalado residente el CPU.
- . Salir de cualquier programa que se este ejecutando ,especialmente copias de versiones anteriores de Matlab
- . Asegurarse de poseer el numero personal de Licencia (PLP).
- . Asegurarse de que las características del equipo utilizado cumpla con los requerimientos mínimos de Matlab

Requerimientos de sistema

- . Procesador Pentium, Pentium Pro, Pentium II, Pentium III, or AMD Athlon.
- . Microsoft Windows 95, Windows 98 (original y Second Edition), Windows NT 4.0 (con Service Pack 5 para cumplimiento de Y2K) o Windows 2000.
- . CD-ROM drive (para la instalación).
- . 64 MB de memoria RAM mínima para Windows 95,98, NT4.0 y 2000, aunque se recomienda especialmente el uso de 128 MB. El espacio libre de disco rígido varia dependiendo del tamaño de la partición y de los archivos de ayuda on-line que se instalen. El software instalador de Mathworks informara durante la instalación si el espacio libre es suficiente.
- . Adaptador grafico y display de 8 bits (para 256 colores simultáneos).

Otros ítem recomendados

- . Tarjeta aceleradora de gráficos
- . Impresora compatible con windows
- . Tarjeta de sonido
- . Microsoft Word 7.0, Office 97 o 2000 es requerido para utilizar el Notebook

Uno de los siguientes softwares es requerido para la construcción de archivos MEX

- . Compaq Visual Fortran 5.0 o 6.1
- . Microsoft Visual C/C++ Versión 5.0 o 6.0
- . Borland C/C++ Version5.0 o 5.02.

Para ver la documentación on-line en formato PDF es necesario instalar el software Acrobat Reader

1.2 Procedimiento de instalación

Durante el proceso de instalación se debe responder tres preguntas básicas:

Quien: Especificar su nombre, el nombre de la compañía y la información de la licencia(PLP)

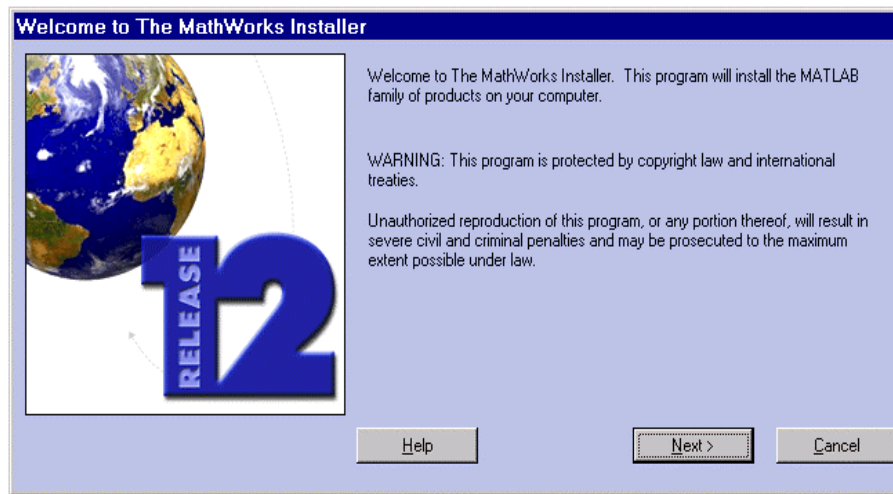
Que: Seleccionar los productos y la documentación que desea instalar

Donde: Elegir el directorio en el cual desea instalar los productos Mathworks

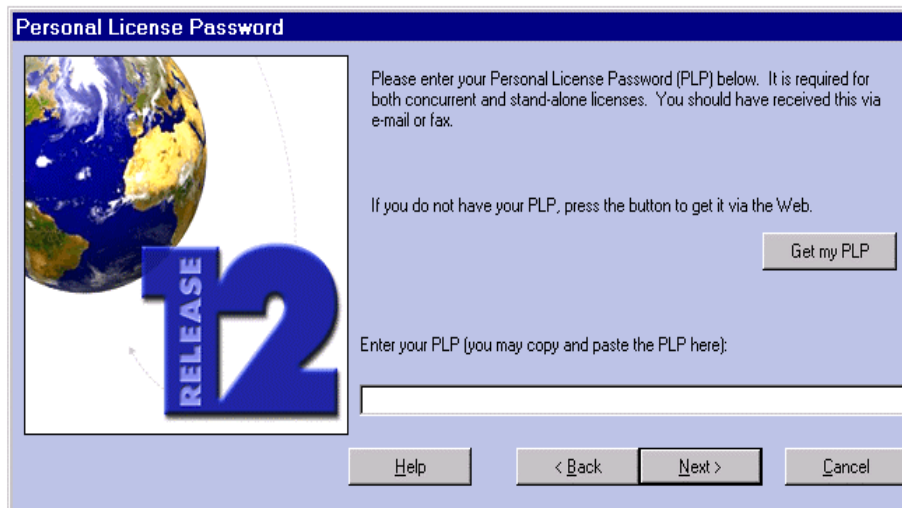
El programa instalador de Mathworks (setup.exe) lo guiara a través del proceso

1.3 Instrucciones de Instalación

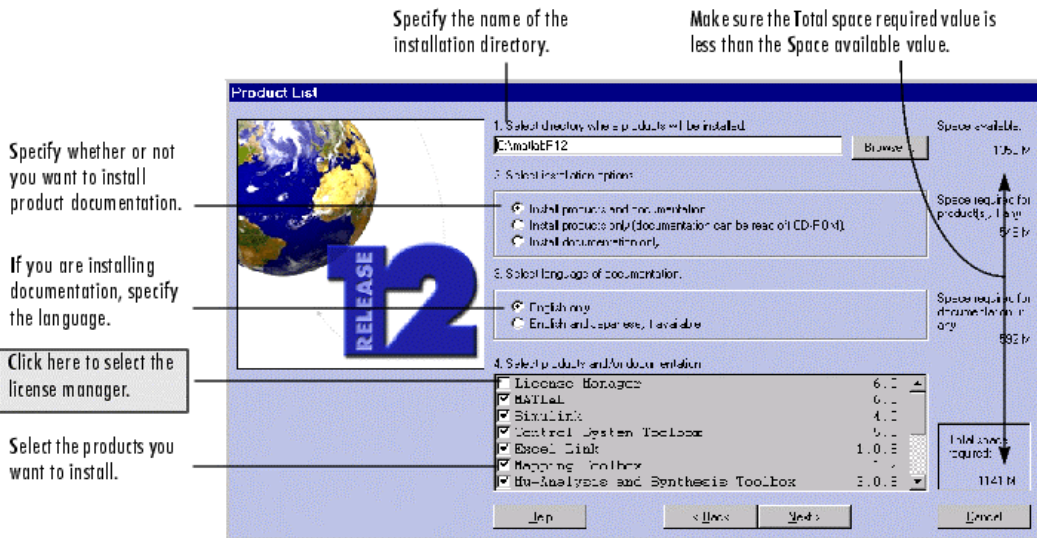
1. Inserte el CD que contiene el producto Mathworks dentro de la lectora de CD ROM y haga doble click en el archivo setup.exe. El instalador arrancará automáticamente mostrando el siguiente cuadro de dialogo de bienvenida



2. Luego de clickear en el botón Next aparecerá el siguiente cuadro de dialogo en el cual deberá ingresar el numero de licencia personal (PLP) .Una vez ingresado haga click en Next y



3. Ingrese su nombre y el nombre de la compañía en le cuadro de dialogo Customer Information y haga click en el botón Next. Aparecera el siguiente cuadro de dialogo Product List
4. Seleccione los productos que desea instalar y el directorio en el cual quiere instalarlos. (Los valores por omisión son mostrados).



5. Haga click en el botón Next para instalar todos los productos y documentación seleccionados en el directorio por omisión (C: \matlabR12). Si desea especificar un directorio diferente tenga en cuenta lo siguiente:

.No utilice la palabra "private" como directorio de instalación

.No utilice un nombre de directorio que contenga espacios o que comience con @.

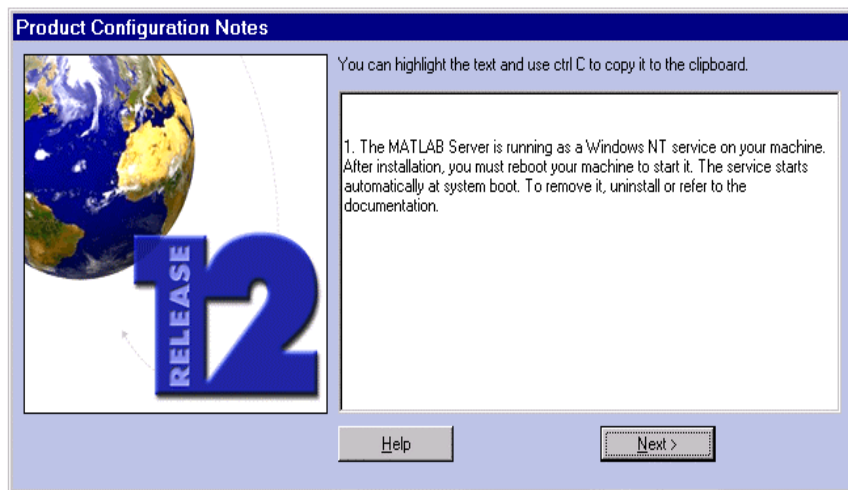
.Si el directorio que especifica no existe el instalador lo crea.

Selección de productos. Si no desea instalar algún producto en especial, quítelo de la selección clickeando en el cuadro de selección junto al nombre del producto. Tenga en cuenta que podrá instalar cualquier producto luego.

Instalando la Documentación. Si elige instalar la documentación al disco , el instalador le indica insertar el CD de documentación en el CD-ROM para completar la instalación. Si por el contrario planea ver la documentación desde el CD , elija la opción de instalar product-only. Si cambia de idea se puede instalar la documentación por separado luego.

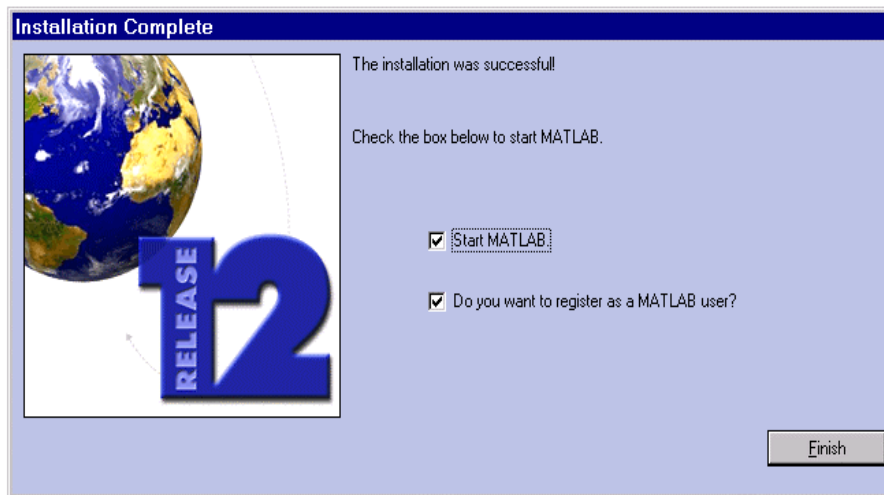
Nota: el instalador no instala la documentación en formato PDF, estos archivos se deben abrir desde el CD para su visualización.

6. Dependiendo de los productos que ha seleccionado el instalador presentara en el siguiente cuadro de dialogo



Se puede copiar información de este cuadro de dialogo para su posterior uso en la configuración.

7. Cuando el instalador finaliza muestra el cuadro de dialogo **Installation Complete**



En este cuadro de dialogo ,se puede elegir iniciar Matlab o registrarse como usuario de Matlab .Si elige lo ultimo se inicia el navegador de internet y abre la pagina de MathWorks Web site para registrar el producto.

Si ha instalado productos que requieran la reiniciación de su sistema antes de ejecutarlos el instalador mostrara esa alternativa en el siguiente cuadro de dialogo.



1.4 Inicio de Matlab

Hay varias maneras de iniciar Matlab:

.Hacer doble click en la icono MATLAB (icono de acceso directo) que el instalador crea en su escritorio.



.Hacer click en el botón de Inicio en el escritorio, seleccionar Programas, y clickear en la entrada MATLAB Release 12. De este menú seleccionar MATLAB R12 para iniciar Matlab.

.El instalador también crea un icono de acceso directo del ejecutable Matlab en el nivel superior del directorio de instalación ,pudiendo ejecutarse también desde dicho lugar.

1.5 Como terminar Matlab

Para salir de Matlab en cualquier momento haga lo siguiente :

.Seleccione el cuadro de cerrar en el escritorio de Matlab.

.Seleccione Exit MATLAB desde el menú File en el escritorio.

.Presione las teclas Ctrl+Q.

.Escriba "quit" en la línea de comandos de la Ventana de Comandos (Command Window) y presione Enter .

Corriendo un Script cuando sale de Matlab

Cuando Matlab termina este ejecuta el archivo script "finish.m" (es un archivo que contiene una secuencia de ordenes a ejecutar) si el mismo existe en el directorio de instalación o en cualquiera de los caminos de búsqueda de archivos del Matlab.

El usuario crea el archivo finish.m. Este poseerá funciones a correr cuando se cierre el Matlab, como por ejemplo salvar el estado del espacio de trabajo (workspace) o mostrar un cuadro de dialogo de confirmación. Hay dos archivos de muestra que le servirán de guía para crear su propio finish.m, estos se encuentran en el directorio MatlabR12\toolbox\local.

Los archivos mencionados son:

finshsav.m, que incluye una función Save de manera que todo el *workspace* es grabado a un archivo MAT cuando Matlab sale.

finishdlg.m, muestra un dialogo de confirmación que permita cancelar la salida de Matlab.

1.6 Estructura de Directorios

Directorio predeterminado de Trabajo

Es el directorio donde se almacenaran las variables del workspace y los archivos creados.

Cuando se inicia Matlab usando el acceso directo del escritorio , el directorio de trabajo inicial será MATLABR12\work (MATLABR12 es el directorio de instalación si no especifico otro durante la misma). Sin embargo se puede utilizar cualquier directorio como el directorio predeterminado de inicio.

De todos modos no se recomienda cambiar el directorio de trabajo work ya que en el se guardan los archivos M que usted a creado o modificado y estos no serán sobrescritos por una nueva instalación o actualización . Así mismo este directorio no será borrado si ejecuta el desinstalador de Matlab

Modificación del Directorio predeterminado de Trabajo

Para especificar un directorio diferente al inicial, debe clickear sobre el icono de acceso director que el instalador creo en el escritorio y seleccionar la opción Properties. Luego especifique el nombre del directorio en el campo Start

1.7 Uso del Help

Las principales formas para acceder a la ayuda en línea son:

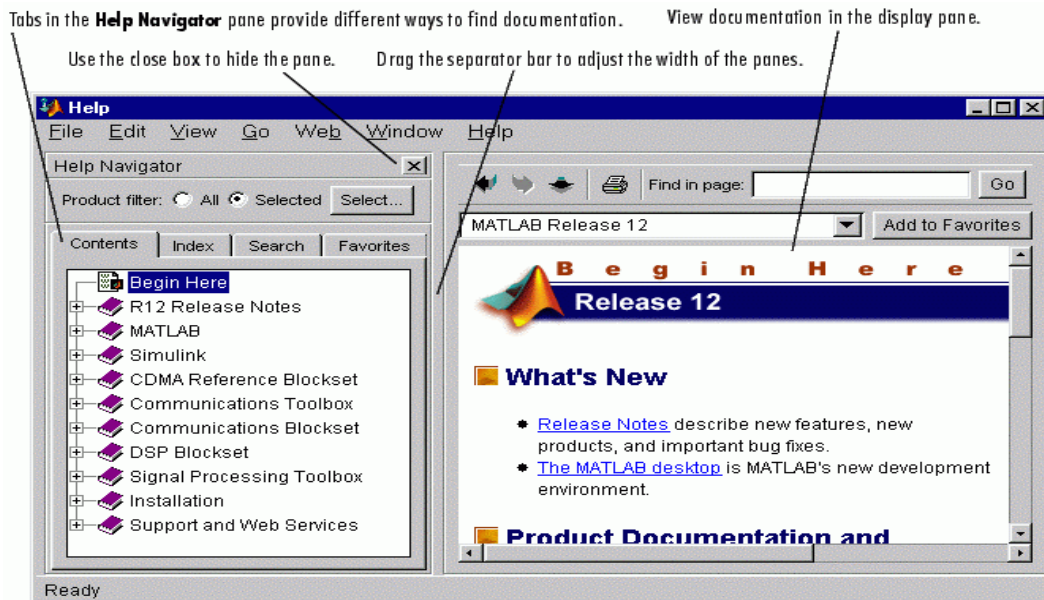
.Usar el **Help Browser**, use el Help browser para encontrar la información que necesita. El mismo cuenta con listados por contenidos, índice global y búsqueda .

.Usar el **Help Function** (funciones de Ayuda), una de las formas es escribir "help functionname" en el Command Window el cual proporcionara una breve descripción de la función y su sintaxis en el Command Window (ver mas adelante Uso de Help Function).

Además del uso de Help en línea existe la posibilidad de imprimir los manuales y tutoriales que vienen en el CD de instalación.

Uso del Help Browser

Use el Help Browser para buscar y ver documentación de Matlab y otros productos MathWorks. El Help Browser esta integrado en el escritorio de Matlab y muestra documentos tipo HTML. Para abrir el Help Browser se debe hacer click en el botón de help de la barra de tareas o escribir helpbrowser en la línea de comandos de la Command Windows y luego presionar Enter. También puede acceder al Help Browser seleccionando Help del menú View en el escritorio de Matlab. Una vez abierto aparecerá el siguiente cuadro de dialogo del Help Browser.



El Help Browser consiste de dos paneles:

.El help Navigator en la izquierda, el cual se usa para encontrar la información. El mismo consta de la posibilidad de Filtrar por productos(Product Filter) y las orejetas de Índice(Index), Búsqueda (Search), Contenidos(Contents) y Favoritos(Favorites)

.El panel visualizador en la derecha , el cual se utiliza para ver la documentación en formato HTML.

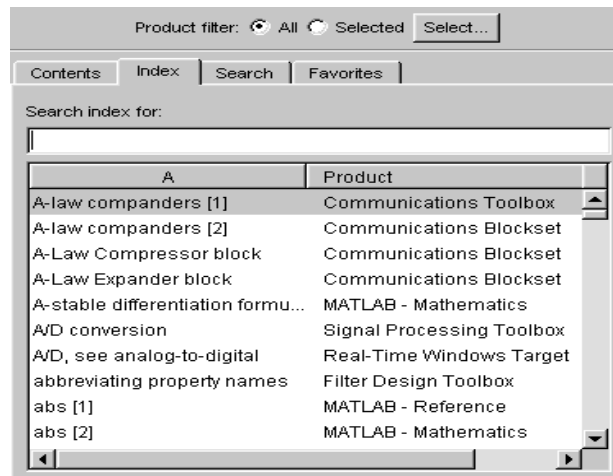
Uso del Help Navigator

El Help Navigator permite buscar información de las siguientes maneras:

.Product filter: Se utiliza para encontrar información solo referente al producto de MathWorks en el que este interesado. Si desea buscar en toda la documentación debe seleccionar All. Para mostrar un subconjunto de productos instalados en el sistema debe clickear en en el botón Select.

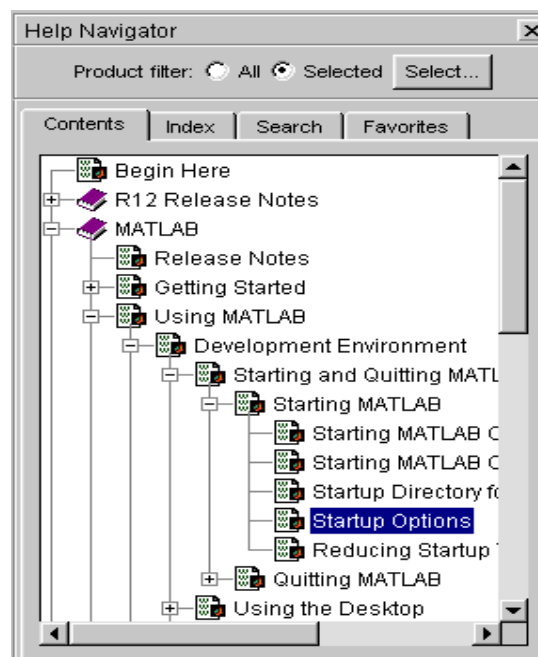
Una vez seleccionado el producto clicke en el Help en el botón Selected lo cual dará el siguiente resultado: Los Contenidos, el Índice y la Búsqueda sollo mostraran información contenida en dicho subconjunto.

.Index: Permite encontrar información del texto ingresado en la entrada de la orejeta Index en el Help Navigator . Para encontrar información escriba en la entrada de texto y a medida que va escribiendo el índice se va modificando para coincidir con su texto. El nombre de un producto o documento que incluye el texto que ingreso aparece después del índice . Puede tomar unos momentos el actualizar el índice. Lo cual es muy útil cuando hay muchas concordancias con el texto ingresado. ver siguiente figura.

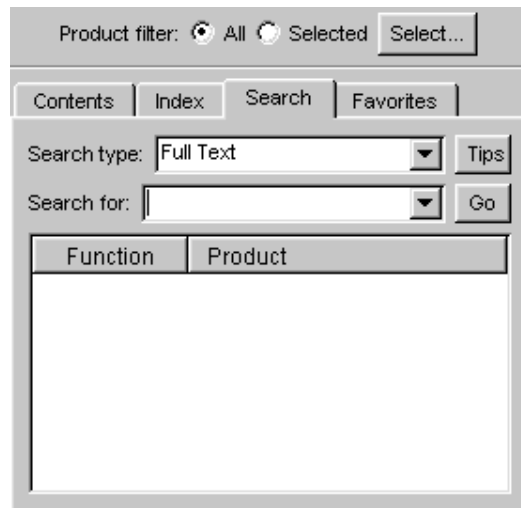


Consejos para el uso del Index: Haga el panel mas ancho para poder ver el producto o el documento al cual el texto ingresado pertenece, escriba en diferente orden las palabras ingresadas en la búsqueda, si el Product filter esta en Selected pruebe volviendo a All.

.Contents: Clickeando en la orejeta de contenidos se puede ver un índice de la documentación por producto. Clickeando en el signo + de cualquier documento se expande el índice de dicho documento, luego si hacemos doble click en el ítem seleccionado nos muestra la primera pagina del documento. En la siguiente figura podemos ver la estructura de documentos.



.Search: Se utiliza para encontrar un frase especifica dentro de la documentación, se selecciona la orejeta de search en el help Navigator y se indica el tipo de búsqueda que se quiere realizar, Full Text busca a través de todos los textos de la documentación en línea de Matlab, Function Name busca en las de referencia de las funciones, Online Knowledge Base se conecta al sitio de Internet de Matlab para encontrar información técnica de soporte y Document Title busca el texto ingresado en los títulos de los documentos. Para iniciar una busqueda, seleccione Search Type, escriba la o las palabras en el cuadro Search for y haga click en el cuadro Go. Ver la siguiente figura



Uso del Help Function

Las funciones de ayudas son ordenes (comandos) que se escriben en la línea de comando del Command Window y son una forma practica y rápida de obtener información acerca de funciones y temas relacionados . En el entorno Windows podemos utilizar los siguientes comandos de ayuda:

.doc: muestra la pagina de referencia (idem Help Browser) de la función especificada luego de este comando .Provee la sintaxis, ejemplos y enlaces a otras funciones directamente relacionadas.

.help: este comando muestra una breve descripción y su sintaxis de la función especificada en el Command Windows .

.helpbrowser: Inicia el help Browser mencionado anteriormente.

.helpwin : muestra en el Help Browser una lista de todas la funciones.

.lookfor: muestra en el Command Window un lista y una breve descripción de todas la funciones incluidas en el texto siguiente al comando.

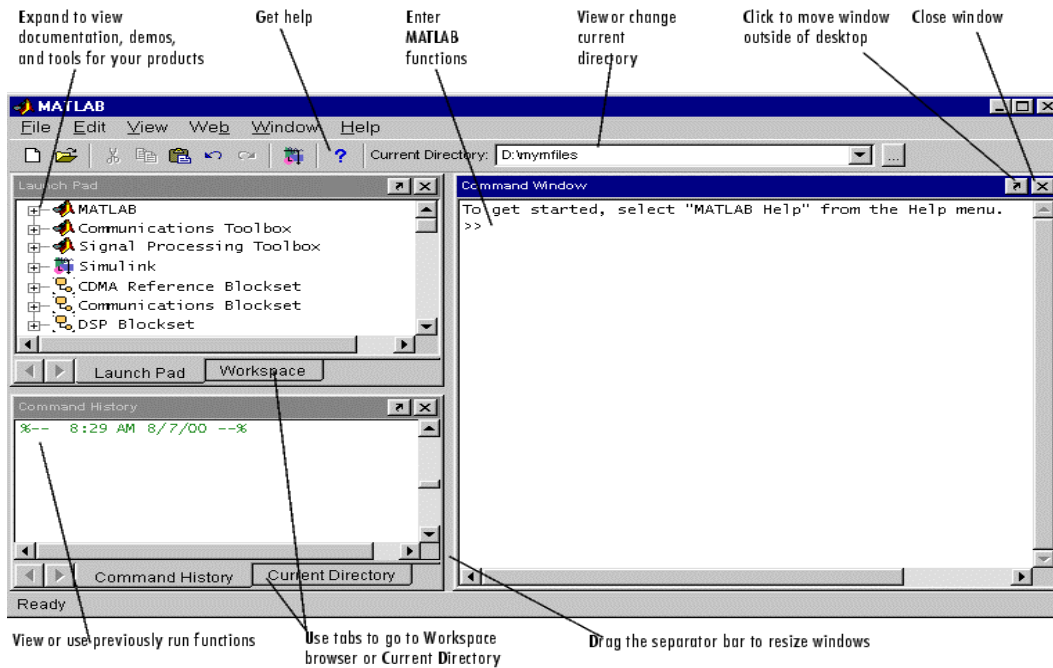
.web: abre una dirección URL en el Web Browser especificado. Se puede utilizar para mostrar documentación en los archivos M.

2.Descripción del Ambiente de Trabajo

2.1. Escritorio de Matlab

Cuando se inicia Matlab aparece el escritorio de trabajo , conteniendo las herramientas para manejar archivos, variables y ejecutar comandos asociados con Matlab. La primera vez que se inicia Matlab el escritorio aparecerá como la siguiente figura , aunque el Launch Pad puede ser diferente de acorde a los componentes que halla elegido durante la instalación. Se puede cambiar la configuración del escritorio abriendo, cerrando , cambiando el tamaño o moviendo las ventanas de las diferentes herramientas.

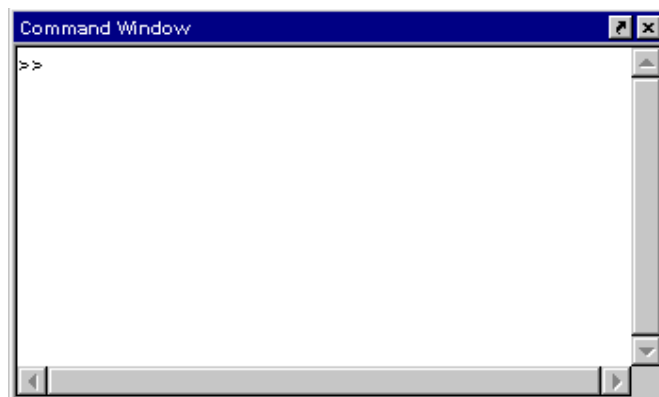
Para especificar ciertas características de las herramientas del escritorio se debe entrar a Preferences en el menú File del escritorio. Las herraminetas del escritorio son: Command Window, Command History, Launch Pad, Help Browser, Current Directory Browser, Workspace Browser, Array Editor y el Editor Debugger. A continuación daremos una breve descripción de las mismas.



2.2 Herramientas del Escritorio

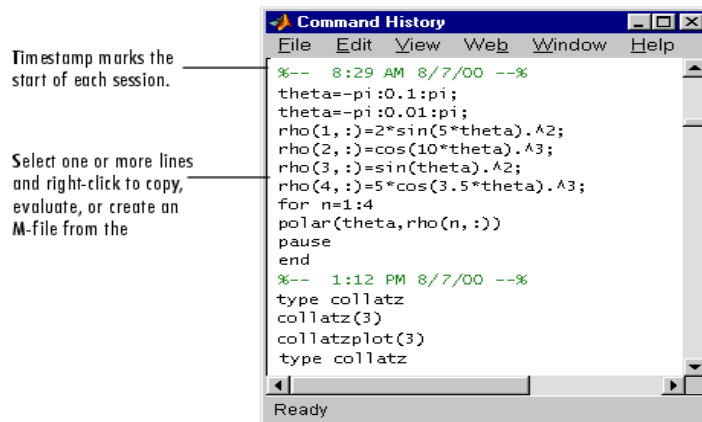
2.2.1. Command Window

La ventana de comandos (Command Windows) es el lugar donde se ingresan las variables, se ejecutan funciones y se corren los archivos M. La siguiente figura como se ve la ventana de comandos en el escritorio. Los comandos se escriben a continuación de >> y se ejecutan con la tecla Enter.



2.2.2. Command History

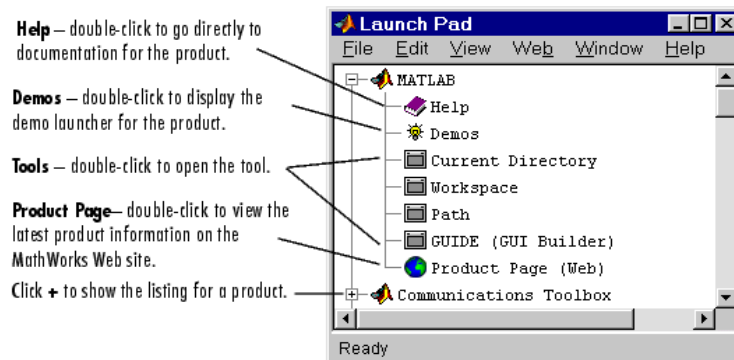
En esta ventana se van guardando los comandos ejecutados en la ventana de comandos. Entonces es muy útil para visualizar funciones ejecutadas anteriormente, lo único que se necesita hacer es seleccionar la función con el mouse y hacer doble click en la misma para que la función o comando se ejecute nuevamente en la ventana de comandos.



2.2.3. Launch Pad

Provee un acceso rápido y fácil a las herramientas, demos y documentación instaladas. Ver figura como ejemplo de uso.

Sample of listings in Launch Pad – you'll see listings for all products installed on your system.



2.2.4. Help Browser

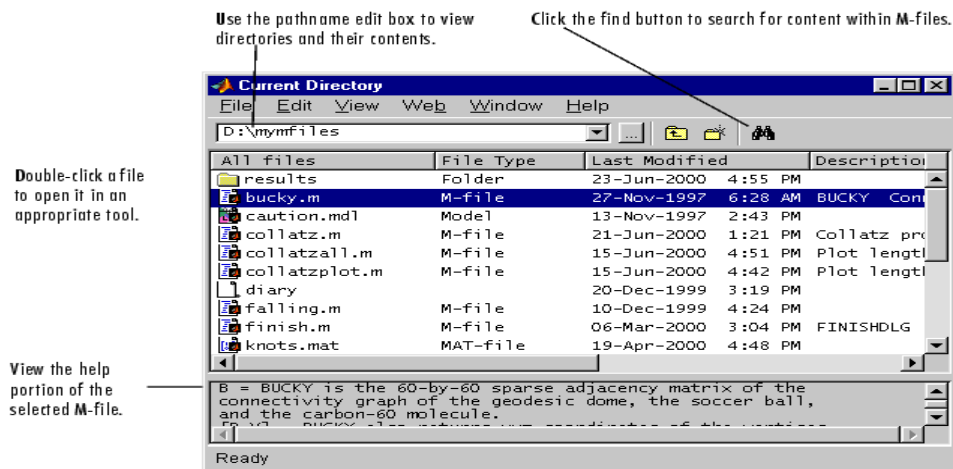
Ver sección 1.7 de este apunte.

2.2.5. Current Directory Browser

Las operaciones con archivos en Matlab usan el directorio corriente (current directory) y el camino de búsqueda (search path) como puntos de referencia. cualquier archivo que se desea ejecutar debe estar en el directorio corriente o en el camino de búsqueda. Una forma rapida para cambiar el directorio corriente es usando el campo Current Directory en la barra de tareas del escritorio , ver la siguiente figura.



Para buscar , ver, abrir o hacer cambios en los archivos relacionados con Matlab se debe utilizar el Current Directory Browser. Alternativamente también se pueden utilizar los comandos *dir*, *cd* y *delete* desde la ventana de comandos, de todos modos es mas simple utilizar el browser por lo que se recomienda tenerlo siempre abierto en el escritorio de trabajo, para ello ingrese en el menú View en el escritorio, seleccione Desktop Layout y seleccione Five Panel. el escritorio de trabajo se configurara de modo que se vean el Command Window, Command History, Launch Pad , Workspace y el Current Directory Browser.



Search Path (Camino de búsqueda) : Cuando ejecuta una función o un archivo Matlab utiliza el camino de búsqueda para encontrar los archivos M y otros relacionados con Matlab. Cualquier archivo que se desee ejecutar debe estar en el *directorio corriente* o en un directorio que este incluido en el *camino de búsqueda*. Durante la instalación todos los archivos y funciones suministradas con Matlab son incluidas en el camino de búsqueda.

Para ver cuales son los directorios predeterminados o cambiar el camino de búsqueda debe seleccionar Set Path desde el menú File en el escritorio de Matlab.

Alternativamente desde la ventana de comandos puede utilizar la función *search* para ver el camino de búsqueda, la función *addpath* para agregar directorios al camino de búsqueda y *rmpath* para quitar directorios del camino de búsqueda.

2.2.6. Workspace Browser

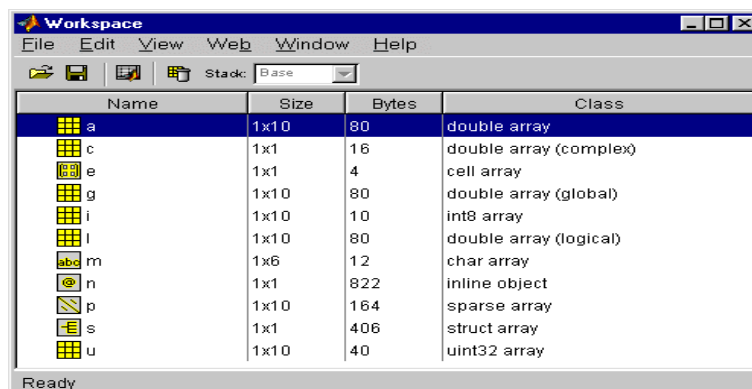
El *workspace* de Matlab consiste en un conjunto de variables (llamadas *arrays*) que son construidas y almacenadas en memoria durante una sesión de Matlab. Las variables se agregan al *workspace* mediante el uso de funciones, corriendo archivos M o cargando *workspaces* grabados en disco.

Para ver las variables del workspace así como también información de las mismas se puede utilizar el Workspace Browser (perteneciente al escritorio de Matlab) o ejecutar las funciones *who* y *whos* desde la línea de comandos de la ventana de comandos.

Para borrar variables del *workspace* seleccione la variable y luego seleccione Delete del menú Edit.

Alternativamente utilice la función *clear* desde la línea de comandos

El *workspace* no es mantenido luego de cerrar una sesión, para grabarlo a un archivo selecciona Save Workspace As desde el menú File. Alternativamente puede utilizar la función *save* desde la línea de comandos. De este modo se graba el Workspace a un archivo binario cuya extensión es *.mat*. Hay opciones para guardarlo de diferentes formatos.



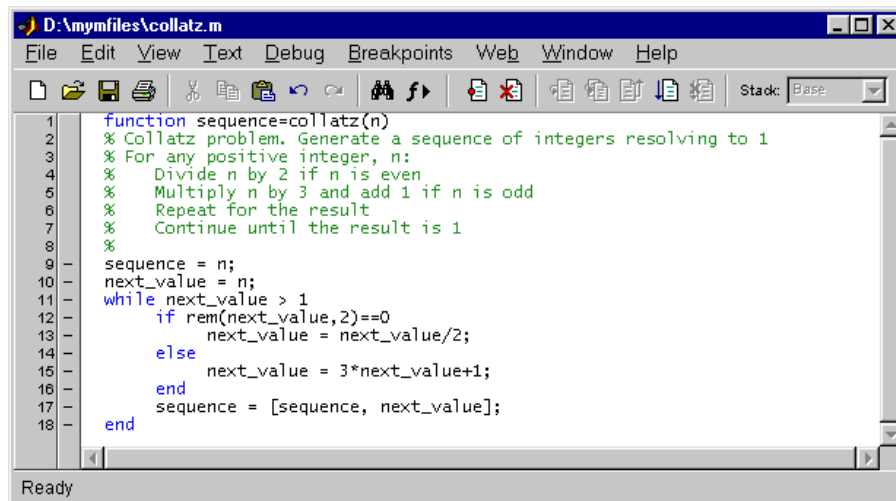
2.2.7. Editor/Debugger

Esta herramienta se utiliza para crear o depurar archivos M, los cuales son programas que usted utiliza para correr funciones Matlab. Este editor provee una interfaz grafica para edición de texto básica y a su vez opciones de depuración que le permite correr el programa paso a paso ,por ejemplo , y poder así detectar un error de ejecución o funcionamiento de un programa.

Se puede utilizar cualquier editor de texto para crear archivos M, de todos modos por simplicidad y practicidad se recomienda utilizar el que viene con Matlab.

Una forma rápida de ver los contenidos de un archivo M es usando la función *type* desde la ventana de comandos, entonces el listado del programa aparecerá en la ventana de comandos para su inspección.

La interfaz grafica se ve de la siguiente manera:



```
D:\mymfiles\collatz.m
File Edit View Text Debug Breakpoints Web Window Help
[Icons] Stack: Base
1 function sequence=collatz(n)
2 % Collatz problem. Generate a sequence of integers resolving to 1
3 % For any positive integer, n:
4 %   Divide n by 2 if n is even
5 %   Multiply n by 3 and add 1 if n is odd
6 %   Repeat for the result
7 %   Continue until the result is 1
8 %
9 sequence = n;
10 next_value = n;
11 while next_value > 1
12     if rem(next_value,2)==0
13         next_value = next_value/2;
14     else
15         next_value = 3*next_value+1;
16     end
17     sequence = [sequence, next_value];
18 end
Ready
```

2.3. Ejecución de Funciones

2.3.1. Ejecución de Funciones e ingreso de variables

Las funciones o comandos de Matlab se ejecutan en la ventana de comandos, una vez abierta esta ultima el prompt `>>` aparecerá para indicarle que esta lista para ejecutar comandos y funciones de Matlab.

.Para ingresar una **variable** al *workspace* , por ejemplo la siguiente línea crea una matriz

A que se almacena durante la sesión en el *workspace*:

```
A = [1 2 3; 4 5 6; 7 8 10]
```

Luego de ingresar todo el texto presionamos Enter y el Matlab devuelve :

```
A =
     1     2     3
     4     5     6
     7     8    10
```

.Para ejecutar una **función** , en la línea de comandos se debe escribir el nombra de la función junto con los argumentos que esta necesite y presionar Enter , entonces Matlab presentara el resultado de dicha ejecución .Por ejemplo el comando:

```
magic(2)
```

retorna como resultado luego de presionar Enter

```
ans =
     1     3
     4     2
```

Si se necesita ingresar múltiples líneas de comando antes de ejecutar de debe usar Shift+Enter después de cada línea de comando hasta la ultima y luego presionar Enter para ejecutar todas las líneas de comando

Si el prompt en la ventana de comandos es `K>>` indica que estamos en modo Depuración.

Para obtener mas información acerca de todas las funciones que provee Matlab debe abrir el Help

Navigator y en la orejeta Contents seleccionar la carpeta MATLAB/Reference/Matlab Functions Reference , una vez ahí podrá buscar las funciones por categoría o por orden alfabético.

Nota: Matlab es sensible a las letras mayúsculas, que quiere decir esto, por ejemplo la función *plot* y *Plot* no son la misma para Matlab. Lo mismo ocurre con las variables por ejemplo una variable llamada *a* no es la misma que *A*.

2.3.2.Ejecución de múltiples funciones

Para ingresar múltiples funciones en una sola línea se deben separar las mismas con un punto y coma (;) por ejemplo podemos ingresar las siguientes tres funciones y las mismas se ejecutaran de izquierda a derecha una vez que presionamos Enter.

2.3.3.Ejecución de Funciones largas

Si el enunciado de una función no entra en una línea se puede seguir en la línea siguiente terminando la función con tres puntos seguidos, luego se presiona Enter para seguir en la otra línea, por ejemplo veamos la siguiente función:

$$s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 \dots \\ - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;$$

2.3.4.Colores del Texto

Algunas palabras en el texto de una función o programa pueden aparecer en diferentes colores:

Luego del signo % el texto que le sigue aparecerá en verde indicando que es un comentario.

Cuando se comienza a escribir un *string* aparecerá en violeta hasta que se cierre con un comilla, luego de eso pasa a rojo.

Cuando se escriben las palabras claves de control de flujo aparecen en color azul.

2.3.5.Como limpiar la ventana de comandos

Para limpiar la ventana de comandos debe seleccionar Clear del menú Edit del escritorio, esta función limpia solamente la vista de la ventana de comandos pero no el workspace.

Equivalentemente se puede utilizar el comando *clc* desde la línea de comandos para realizar la misma operación.

2.3.6.Eliminando salidas de resultado

Cuando se termina una línea de comando con el símbolo punto y coma (;), el comando se ejecuta pero el resultado no se presenta en pantalla luego de presionar Enter. Esta particularidad es muy útil cuando trabajamos por ejemplo con matrices grandes cuya presentación excede las dimensiones de la pantalla.

2.3.7.Paginación de la ventana de comandos

Si el resultado (salida) que se presenta en la ventana de comandos es muy larga (por ejemplo operaciones con matrices y vectores de gran longitud) , en la misma se verán las últimas líneas del resultado perdiéndose en ciertos casos la vista de algunos datos. Para evitar esto exista la función *more* para controlar el paginado de la ventana de comandos. Cuando se escribe *more on* en la línea de comandos Matlab muestra una página de resultados a la vez. Una vez que la primera pantalla se presenta debe presionar *Enter* para avanzar hacia la próxima línea, *Space Bar* para avanzar a la próxima página o la letra *q* para salir.

2.3.8.Ejecución de Programas

Los archivos M ,archivos que contienen código en el lenguaje Matlab, se corren de la misma manera que se corren las funciones. Se escribe el nombre en la línea de comandos y se presiona Enter. Para ver cada función que se va ejecutando en el archivo utilice la función *echo* en *on* (escriba *help echo* para más detalles).

Para interrumpir la ejecución de un programa presione Ctrl+c o Ctrl+Break en cualquier momento de la ejecución y el programa se detendrá.

2.3.9.Guardar sesiones de Trabajo

La función *diary* crea una copia en disco de la sesión de Matlab, incluyendo entradas del teclado y repuestas del sistema, aunque no incluye gráficos. Para iniciar la grabación de una sesión debe escribir en la línea de comandos:

```
diary('name.out')
```

donde *name* es el nombre del archivo de texto donde se guardaran las entradas y salidas de su sesión.

Para terminar de grabar la sesión debe escribir:

```
diary('off')
```

2.4.Editor de Arreglos y Matrices

2.4.1.Diferencia entre arreglos y matrices

Las matriz es un arreglo numérico en el cual los datos (números) se ordenan por fila y columna. Un elemento de la matriz *A* será referido como el elemento $A(i,j)$ donde *i* es el número de fila a la que pertenece y *j* la columna a la que pertenece. Un arreglo propiamente dicho (Array) puede estar compuesto por otro tipo de datos además del numéricos. La diferencia entre un array y una matriz cuando ambos están formados por números son las operaciones matemáticas que Matlab define sobre ellos (excepto suma y resta donde las operaciones son iguale para matrices y arrays) las demas operaciones en el caso de las matrices están gobernadas por el álgebra lineal mientras que para los arreglos las operaciones se definen elemento a elemento. En la seccion 2.5 veremos en detalles estas operaciones .

2.4.2.Editor

Una forma rápida de editar arreglos unidimensionales y bidimensionales (matrices) del workspace es usando el Array Editor , los arreglos pueden ser numéricos o cadenas (strings) de caracteres.

Básicamente es un cuadro de dialogo donde se presentan todas las variables de workspace y se pueden editar visualmente.

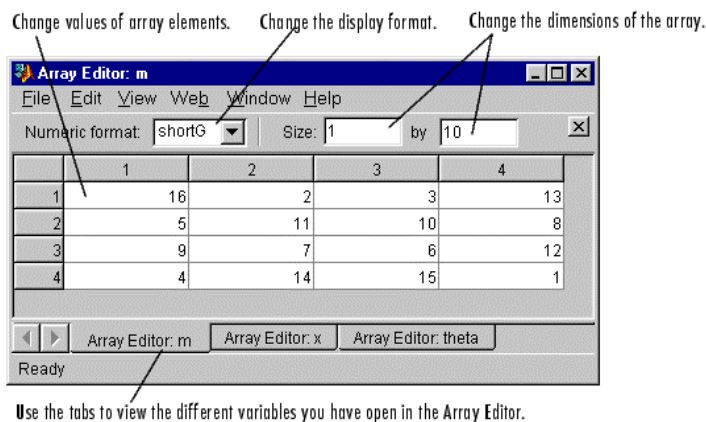
El editor de arreglos se abre desde el Workspace browser . Una vez en este se selecta la variable o múltiples variables a editar mediante Ctrl+click o Shift+click y luego hace click derecho del mouse y seleccionar Open Selection.

Una forma alternativa es utilizar la función *openvar* en la línea de comandos. Por ejemplo veamos la siguiente sintaxis:

```
openvar('m')
```

Matlab abrirá la variable *m* en el Array Editor

La siguiente figura muestra el cuadro de dialogo y componentes del Array Editor



2.5 Importación y Exportación de Datos

Matlab provee diferentes maneras para cargar datos desde archivos de disco o del *clipboard* hacia el workspace, estos procesos se llaman importación de datos. También puede grabar datos (variables del workspace) a archivos de disco, este proceso se llama exportación de datos. La elección del mecanismo a usar depende de la operación que este realizando, importar o exportar, y el formato del dato, binario o texto.

Nota: La forma mas fácil de importar datos hacia el Matlab es usando el Import Wizard. Cuando se utiliza esta herramienta no necesita saber el formato del dato a importar. Basta simplemente con especificar el nombre del archivo y el Import Wizard procesa el contenido del archivo automáticamente. Para mas información busque en el Help Using the Import Wizard with Text Data and Binary Data Files.

Datos en formato de Texto

En el formato texto los valores de los datos son los códigos del American Standard Code for Information Interchange (ASCII) que representan caracteres y valores numéricos. Un texto de datos ASCII puede ser visto en cualquier editor de texto.

2.5.1. Importación de datos en formato texto: Hay dos maneras de importar datos, una es a través del ya mencionado Import Wizard y la otra es desde la línea de comandos utilizando las funciones específicas de importación. En este ultimo caso se debe tener la precaución del formato del dato en el archivo pues del el dependerá la función que se utilice.

Nota: Cuando importa datos dentro del workspace cualquier variable que tenga el mismo nombre de una importada será sobrescrita por esta ultima.

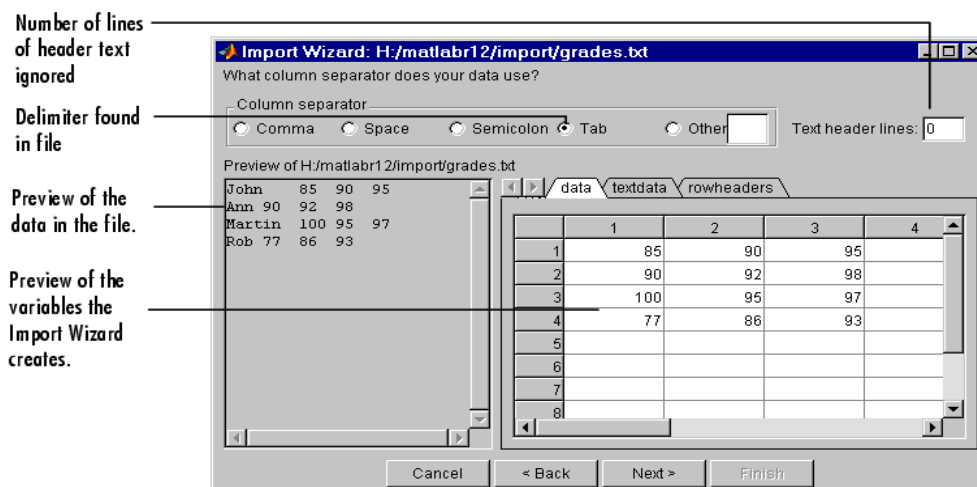
.Uso del Import Wizard: Se debe seleccionar la opción Import Data del menú File y entonces se abrirá un cuadro de dialogo que le permite seleccionar el archivo a importar. Una vez seleccionado se debe clicar en Open y automáticamente se abre el Import Wizard listo para procesar el contenido del archivo. Se debe especificar el carácter para separar los datos individuales. Este carácter es llamado delimitador o separador de columna.

Seleccione las variables a importar , por defecto el Import Wizard pone todos los datos numéricos en una variable y todos los datos de texto en otra.

Por ejemplo si abriéramos el siguiente archivo (grades.txt) conteniendo el siguiente texto

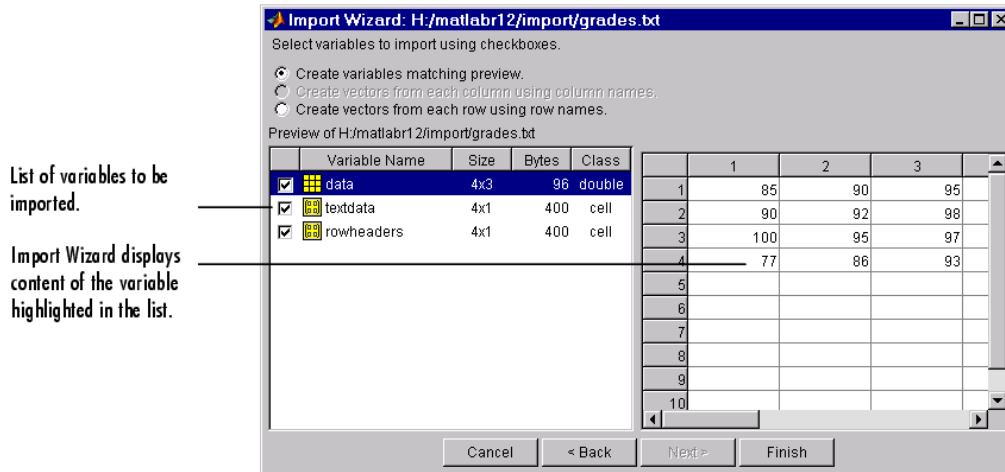
```
John 85 90 95
Ann 90 92 98
Martin 100 95 97
Rob 77 86 93
```

En el Import wizard se vería de la siguiente manera :



En la figura se debe notar como se auto detectó el separador de columnas que en este caso es Tab, también se puede ver que se crearon tres tipos de datos : tipo *data*, conteniendo los datos numéricos , *textdata* conteniendo todo el texto encontrado y *rowheader* conteniendo los nombres de la columna izquierda de *data*.

Para seleccionar la variable a importar debe clickear en Next en el cuadro anterior y para elegir la variable en el siguiente cuadro de dialogo debe seleccionar en cuadro de selección junto a la variable.



.Uso de la Funciones de Importar: Se importan datos desde la línea de comandos. La funcion elegida depende de como los datos en el archivo de texto estén formateados. El texto debe estar formateado en patrones uniformes de filas y columnas usando un separador (delimitador) que puede ser un espacio, una coma, tab , punto y coma o cualquier otro carácter .

El texto puede contener una o mas líneas de texto llamadas líneas de encabezado. En el siguiente ejemplo se pueden ver un archivo de texto delimitado por Tab con encabezado de texto y con encabezados de columnas y filas.

Archivo de texto con sus diferentes componentes

Text header line	Class Grades for Spring Term			
Column Headers	Grade1	Grade2	Grade3	
Row Headers	John	85	90	95
	Ann	90	92	98
	Martin	100	95	97
	Rob	77	86	93
Tab-delimited data				

En la siguiente tabla daremos un breve resumen de los tipos de datos y las funciones que se utilizan para importarlos.

Función	Tipo de dato	Valores de Retorno	Delimitadores	Notas
<i>textread</i>	Alfabetico y Numérico	Múltiples	Cualquier caracter	Flexible, potente y fácil de usar
<i>csvread</i>	Numérico	Uno	Solo comas	Carga de datos
<i>dlmread</i>	Numérico	Uno	Cualquier caracter	Flexible y fácil de usar
<i>fscanf</i>	Afbabetico y Numérico	Uno	Cualquier caracter	Rutinas de bajo nivel, se usa en conjunto con <i>fopen</i> y <i>fclose</i>
<i>load</i>	Numérico	Uno	Solo espacios	Fácil de usar desde el command windows

Se puede utilizar el help en la línea de comandos para ver detalles de las funciones mencionadas anteriormente.

2.5.2.Importación de Datos Binarios: La forma mas fácil es utilizando el Import Wizard anteriormente mencionado. Debido a la similitud con el párrafo anterior describiremos los pasos a seguir. Ahora si necesitamos trabajar desde la ventana de comandos debemos utilizar una de las siguientes funciones provistas por Matlab.

Figura 2.5.2

Formato del Dato	Extensión del Archivo	Descripción y función a utilizar
Audio	.wav	Se utiliza la función <i>wavread</i> para importar archivos de audio
Audio y Video Intercalado	.avi	Se utiliza la función <i>aviread</i> para importar archivos de audio y video intercalado (AVI)
Archivos de imagen	.jpg, .tiff, .bmp, .gif, .png, .hdf, .pck, .xwd	se utiliza la función <i>imread</i> para importar imágenes
archivos M	.mat	Se utiliza la función <i>load</i> para archivos con formato propietario de Matlab
Hoja de Calculo	.xls	se utiliza la función <i>xlsread</i> para importar hojas de calculo Excel

2.5.3 Exportación de Datos ASCII: Daremos a continuación en la siguiente Tabla un guía rápida para el uso de funciones de exportación de acuerdo a los diferentes tipos de datos.

Función	Usar con datos	Delimitadores	Notas
<i>diary</i>	Numérico o arreglos de celdas	solamente espacios	Se puede utilizar en pequeños arreglos. requiere la edición de los datos para quitar caracteres extraños
<i>dlmwrite</i>	Numérico	Cualquier caracter	Fácil y flexible de usar
<i>fprintf</i>	Alfabético y numérico	Cualquier caracter	Rutinas de entrada salida de bajo nivel. se utiliza en conjunto con <i>fopen</i> y <i>fclose</i>
<i>save</i>	Numérico	Tab y espacio	Los valores de salida son de alta precisión

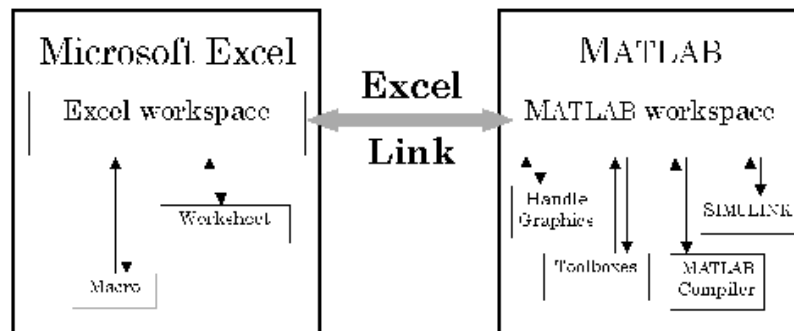
2.5.4 Exportación de Datos Binarios: Nuevamente al igual que la importación el método mas fácil es utilizando el Import Wizard. De igual manera podemos exportar desde la ventana de

comandos. La siguiente Tabla es una guía rápida de las funciones utilizadas. Para detalles de uso use el comando help "nombrefuncion" en la ventana de comandos.

Formato del Dato	Extensión del Archivo	Descripción y función a utilizar
Audio	.wav	Se utiliza la función <i>wavwrite</i> para exportar archivos de audio
Audio y Video Intercalado	.avi	Se utiliza la función <i>avifile</i> junto con <i>addframe</i> y <i>close</i> para importar archivos de audio y video intercalado (AVI)
Archivos de imagen	.jpeg, .tiff, .bmp, .gif, .png, .hdf, .pck, .xwd	se utiliza la función <i>imwrite</i> para exportar imágenes
archivos M	.mat	Se utiliza la función <i>save</i> para grabar archivos con formato propietario de Matlab
Hoja de Calculo	.xls .wkl	se utiliza la función <i>wklwrite</i> para exportar hojas de calculo Lotus 123.

2.6.Excel Link

Excel Link es un Software que integra Microsoft Excel y Matlab en un entorno computacional Windows. Conectando Excel con Matlab se puede acceder al potente procesamiento numérico y grafico de Matlab desde una planilla de calculo. A su vez Excel permite el intercambio de datos entre los dos ambientes. Desde el punto de vista de importación y exportación de datos es una herramienta que permite obtener y enviar datos del Excel a Matlab y viceversa , pero además provee funciones que permiten ejecutar acciones sobre dichos datos y acceder a facilidades de calculo complejo y graficas especializadas del Matlab , resultando una configuración de trabajo donde el Excel es la interfaz grafica para los datos y Matlab el motor de calculo. La siguiente grafica permite entender como es el intercambio de información entre los dos ambientes.



Excel Link comunica los espacios de trabajo del Matlab y de Excel a través de un restringido y simple conjunto de funciones básicas. Se pueden ejecutar la funciones Excel Link desde la Hoja de calculo o desde macros sin tener que dejar el ambiente de Excel . Solo con 11 funciones la potencia de Excel link reside en su simplicidad.

2.6.1. Instalación de Excel Link

Para poder utilizar el Excel Link este debe haber sido seleccionado de la lista de productos durante la instalación de Matlab. Una vez instalado se necesitan realizar los siguientes pasos para configurar el Excel :

- 1) Iniciar Excel
- 2) Seleccionar del menú Herramientas la opción Complementos y luego Examinar
- 3) Encontrar la ruta del EXCLLINK.XLA dentro del directorio C:\Matlabr12\Toolbox\Exlink ,lo

seleccionamos y luego clickeamos OK

4) Volver a la ventana de Complementos en Herramientas y verificar que este tildado el cuadro de verificación e la opción Excel Link.

5) Observar que aparezca el botón de la ventana de Comandos del Matlab en la barra de tareas.

2.6.2. Funciones.

.Funciones de manejo de enlace. son las funciones que permiten inicializar y cerrar Excel Link y Matlab:

matlabinit	Inicializa el Excel Link y arranca el proceso de Matlab
MLAutoStart	Inicia el proceso de Matlab automáticamente
MLClose	Termina el proceso de Matlab
MLOpen	Inicia el proceso de Matlab

.Funciones de manejo de datos.

MLAppendMatrix	Crea una matriz en Matlab con datos de una hoja de calculo
MLDeleteMatrix	Borra una matriz en Matlab
MLEvalString	Evalúa un comando en Matlab
MLGetMatrix	Escribe los contenidos de una matriz Matlab en la hoja de calculo
MLGetVar	Escribe los contenidos de una matriz Matlab en una variable Excel VBA (Visual Basic for applications)
MLPutMatrix	Crea una matriz en Matlab con datos de una hoja de calculo
MLPutVar	Crea una matriz Matlab con datos de una variable Excel VBA.

2.6.3. Tipos de Datos

Excel Link solamente maneja arreglos numéricos bidimensionales ,arreglos de caracteres de una dimensión y arreglos de celdas bidimensionales conteniendo solamente cadenas de caracteres. no trabaja con arreglos multidimensionales, estructuras o arreglos de celdas excepto aquellas que contienen solamente cadenas de caracteres.

Ejercicios

1. Con la función *wavread* importar el archivo *chimes.wav* y convertirlo a una matriz del workspace. Separar esta matriz en 2 vectores columnas (canal izq. y derecho) , graficarlos con la función *plot*, y luego grabar los canales independientes con la función *wavwrite* .
2. Crear una tabla numérica en Excel , leerla con la función *xlsread* y convertirla a una matriz en el workspace . Editarla con el editor de arreglos y grabarla con el comando *save*.
3. Crear con el Notepad una tabla que tenga 3 columnas , Nombre, Apellido y Nota, ingresarle 5 a 10 entradas diferentes y con la función *textread* crear las variables Nombre ,Apellido y Nota en el workspace conteniendo las entradas de dicha tabla.

3. Funciones Generales y Matemáticas de Matlab

3.1. Variables

A diferencia de otros lenguajes de programación Matlab no necesita ningún tipo de declaración del tipo o dimensión de la variable a utilizar .cuando Matlab encuentra un nuevo nombre de variable automáticamente crea la variable y ubica la cantidad necesaria de memoria. Si la variable existe, Matlab cambia el contenido y si es necesario ubica la cantidad apropiada de memoria.

Por ejemplo:

```
cant=25
```

crea una matriz de 1x1 llamada *cant* y almacena el valor 25 en una ubicación de memoria.

El nombre de la variable debe comenzar con una letra y puede ser seguido por letras, números o guiones Matlab utiliza los primeros 31 caracteres y el resto los ignora, también distingue entre mayúsculas y minúsculas, por ejemplo *Cant* no es la misma variable que *cant*.

3.2. Números y Representaciones

Matlab usa la notación decimal convencional con la opción de un signo + o - delante del número para indicar si es positivo o negativo.

Todos los números son almacenados internamente usando el formato largo del estándar de punto flotante de la IEEE. Los números de punto flotante tienen la precisión finita de los 16 dígitos decimales y un rango de 10^{-108} a 10^{+308} .

Hay tres tipos de números utilizados en Matlab : *Enteros, Reales y Complejos*.

Los enteros son ingresados sin el punto decimal por ejemplo: $x=10$ será grabada como un *número entero*. Sin embargo si nuestra variable fuera $x=10.01$ es grabada como un *número real*.

Un número complejo en Matlab es representado en forma rectangular es decir su parte real e imaginaria por separado. La parte imaginaria se puede representar ya sea con la letra *i* o *j*. Por ejemplo si escribimos *i* en la ventana de comandos como resultado obtendremos

```
ans =  
0 + 1.0000i
```

Recordemos que la letra *i* que acompaña al número indica que este está a 90 grados respecto del anterior.

Además de los tipos de números mencionados anteriormente Matlab posee tres variables para representar lo que se denomina nonumeros. Estas variables son:

.-Inf (Representación del infinito negativo)

. Inf (Representación del infinito positivo)

.NaN (Significa que no es un número y se asigna a resultados de operaciones como $0.0/0.0$ o $\text{Inf} \times \text{Inf}$)

Nota: Para cambiar el formato de los números en la ventana de comandos se debe ingresar en el menú File y entrar a Preferences para hacer el cambio. La configuración por defecto es llamada *short* lo que implica que se verán cuatro dígitos después del punto decimal. Los cambios también se pueden realizar desde la ventana de comandos, por ejemplo si escribimos en la línea de *format long* los números pasaran a representarse con 14 decimales.

3.3. Operadores Aritméticos

Las expresiones en Matlab usan los operadores aritméticos comunes así como también las reglas de precedencia de cálculo. Por ejemplo la siguiente expresión :

```
resul= 2*(5+4);
```

evalúa primero la suma dentro del paréntesis y luego evalúa la multiplicación.

La siguiente tabla describe los operadores básicos:

+	Suma
-	Resta
/	División
*	Multiplicación
\	División Izquierda (ver calculo con Matrices)
^	Potencia
'	Traspuesta compleja conjugada
()	Orden de evaluación de la expresión

3.4. Operadores Matriciales vs Operadores Elemento a Elemento

Matlab tiene dos tipos de operaciones aritméticas diferentes. Las operaciones aritméticas con Matrices y Vectores y las operaciones aritméticas con Arreglos (Arrays).

Las primeras están definidas por las reglas del álgebra lineal. Mientras que las operaciones con arreglos

son llevada a cabo elemento a elemento.

La distinción entre unas y otras se logra agregando un punto, (.) , que preceda las operaciones para el caso de arreglos. Sin embargo dado que la suma y la resta es igual para matrices que para arreglos no se hace distinción en estos casos.

Para el caso de la suma y la resta no hay diferencia entre operaciones con matrices y arreglos. Sin embargo en las otras si por ejemplo sean A y B dos matrices/arreglos de dimensiones 2X2.

$$A = \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix}$$

$$B = \begin{matrix} 3 & 3 \\ 3 & 3 \end{matrix}$$

El **producto** matricial esta definido por la siguiente operación:

El numero de columnas de la primera debe ser el mismo que el el numero de filas de la segunda:

$$C=A*B$$

$$C = \begin{matrix} 12 & 12 \\ 12 & 12 \end{matrix}$$

$$C(i,j) = \sum_{k=1}^n A(i,k)B(k,j)$$

El **producto** de arreglos (elemento a elemento) A y B deben ser del mismo tamaño:

$$C=A.*B$$

$$C = \begin{matrix} 6 & 6 \\ 6 & 6 \end{matrix}$$

División Matricial o división derecha. B/A es lo mismo que realizar B* inv(A). Por ejemplo B/A para las mismas matrices anteriores:

$$C=B/A$$

$$C = \begin{matrix} 0.5000 & 1.0000 \\ 0.5000 & 1.0000 \end{matrix}$$

División de Arreglos. Es la división elemento a elemento B(i,j) /A(i,j).A y B deben ser del mismo tamaño a menos que uno de ellos sea un escalar. Por ejemplo

$$C=B./A$$

$$C = \begin{matrix} 1.5000 & 1.5000 \\ 1.5000 & 1.5000 \end{matrix}$$

División Izquierda de Matrices. La operación B\A es la misma operación que inv(B)*A. Si B es una matriz de n x n elementos y A un vector columna con n elementos entonces X=B\A es la solución de la ecuación BX=A.

División izquierda de arreglos B.\A es la operación A(i,j).\ B(i,j).Veamos el siguiente ejemplo

$$C=B.\A$$

$$C = \begin{matrix} 0.6667 & 0.6667 \\ 0.6667 & 0.6667 \end{matrix}$$

Potencia de Matrices. La potencia es computada por sucesivas repeticiones de la multiplicación por si misma es decir A^p es multiplicar AxAxA... p veces donde p es un entero . Si potencia p a la cual se

eleva la matriz es un numero negativo primero se invierte la matriz. Por ejemplo:

$$C = A^p \text{ para } p=2$$

$$C =$$

$$\begin{matrix} 8 & 8 \\ 8 & 8 \end{matrix}$$

$$C = A^p \text{ para } p=-2$$

$$C =$$

$$\begin{matrix} \text{Inf} & \text{Inf} \\ \text{Inf} & \text{Inf} \end{matrix}$$

este resultado es debido a que la matriz es singular es decir su Determinante es cero, entonces no tiene inversa.

Potencia de Arreglos. Es la operación $A.^B$ donde los elementos $A(i,j)$ son elevados a la potencia de los elementos $B(i,j)$. En este caso utilizaremos otra matriz B para que quede de manifiesto la operación .

Entonces

$$B =$$

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

$$C = A.^B$$

$$C =$$

$$\begin{matrix} 2 & 4 \\ 8 & 16 \end{matrix}$$

Traspuesta de una Matriz. B' es la traspuesta lineal algebraica de B. Para matrices complejas esta operación realiza la traspuesta conjugada. Veamos los siguientes ejemplos :

$$C = B'$$

$$C =$$

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

Definimos una matriz B compleja

$$B =$$

$$\begin{matrix} 1.0000 + 1.0000i & 2.0000 + 2.0000i \\ 3.0000 - 3.0000i & 4.0000 - 4.0000i \end{matrix}$$

$$C = B'$$

$$C =$$

$$\begin{matrix} 1.0000 - 1.0000i & 3.0000 + 3.0000i \\ 2.0000 - 2.0000i & 4.0000 + 4.0000i \end{matrix}$$

Traspuesta de arreglos. Es la operación B.', traspone el arreglo. Para el caso de elementos complejos no realiza la conjugación. Por ejemplo:

$$C = B.'$$

$$C =$$

$$\begin{matrix} 1.0000 + 1.0000i & 3.0000 - 3.0000i \\ 2.0000 + 2.0000i & 4.0000 - 4.0000i \end{matrix}$$

Vemos que no realiza conjugación , es decir el signo de la parte imaginaria no cambia.

Nota: Todas estas operaciones tienen su respectivas funciones de archivo M talque se pueden también ejecutar de la siguiente manera en la línea de comandos.

Operación	Función M Equivalente
A+B	plus(A,B)

+A	uplus(A)
A-B	minus(A,B)
-A	uminus(A)
A*B	mtimes(A,B)
A.*B	times(A,B)
A/B	mrdivide(A,B)
A./B	rdivide(A,B)
A\B	mldivide(A,B)
A.\B	ldivide(A,B)
A^B	mpower(A,B)
A.^B	power(A,B)
A'	ctranspose(A)
A.'	transpose(A)

Como respuesta a las operaciones descritas anteriormente pueden aparecer los siguientes mensajes de advertencia en la línea de comandos :

.Si en una división de matrices la matriz cuadrada A es singular

Warning: Matrix is singular to working precision.

.Si una división tiene el divisor igual a cero

Warning: Divide by zero.

.Si se encuentra la inversa de una matriz pero no es confiable

Warning: Matrix is close to singular or badly scaled.

Results may be inaccurate. RCOND = xxx

De una división matricial si una matriz no cuadrada es deficiente en su rango

Warning: Rank deficient, rank = xxx tol = xxx

Si una división de matrices retorna una matriz con todos sus elementos de valor Inf, la división de esta última dará como resultados Inf y NaN donde lo crea conveniente.

3.5. Generación de Vectores y Matrices

La forma básica de declarar o ingresar un vector la ventana de comandos es escribiendo el nombre del vector y luego de signo igual se ingresan los elementos del vector entre corchetes y separados por espacios. Por ejemplo:

```
vec=[1 2 3]
```

da como respuesta el siguiente vector fila.

```
vec =
```

```
1 2 3
```

si queremos hacer que el vector quede ordenado en forma columna podemos utilizar el operador ' que realiza la traspuesta de un vector o matriz. Para nuestro ejemplo:

```
vec'
```

```
ans =
```

```
1
```

```
2
```

```
3
```

Para el caso de las matrices se ingresan de la misma forma, separando las filas con un punto y coma. Por ejemplo:

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

genera la siguiente matriz

```
A =
```

```
1 2 3
```

```
4 5 6
```

```
7 8 9
```

Además Matlab también provee cuatro funciones para generar matrices básicas:

zeros : Genera una matriz con todos sus elementos igual a cero.

ones: Genera una matriz con todo sus elementos igual a uno.

rand: Genera una matriz con valores aleatorios uniformemente distribuidos.

randn: Genera una matriz con valores aleatorios normalmente distribuidos.

Por ejemplo:

`Z = zeros(2,4)` (los números dentro del paréntesis indican la cantidad de filas y columnas respectivamente).

```
Z =  
 0  0  0  0  
 0  0  0  0
```

`F = 5*ones(3,3)`

```
F =  
 5  5  5  
 5  5  5  
 5  5  5
```

`R = randn(4,4)`

```
R =  
 1.0668  0.2944 -0.6918 -1.4410  
 0.0593 -1.3362  0.8580  0.5711  
 -0.0956  0.7143  1.2540 -0.3999  
 -0.8323  1.6236 -1.5937  0.6900
```

Otra forma de ingresar una matriz es utilizando el comando **load**, este comando lee archivos binarios conteniendo matrices generadas en una sesión anterior de Matlab o lee archivos de texto conteniendo datos numéricos. El texto en el archivo debe estar organizado como una tabla de números separados por espacios en blancos , con una fila por línea e igual cantidad de elementos por columna.

Por ejemplo fuera de Matlab con cualquier editor de texto escribimos el siguiente texto y lo guardamos

```
16.0  3.0  2.0  13.0  
 5.0  10.0  11.0  8.0  
 9.0  6.0  7.0  12.0  
 4.0  15.0  14.0  1.0
```

y lo salvamos con el nombre *matri.dat* luego con el comando `load` leemos el archivo y crea una variable con el nombre *matri* conteniendo dicha matriz.

También se pueden crear matrices utilizando archivos *M*, los cuales son archivos de texto conteniendo código de Matlab. Se graba archivo con un nombre con terminación *.m* y luego al ejecutar el nombre del archivo en la línea de comandos crea la variable guardada en el archivo.

Nota: Un tipo de vector de mucha utilidad es aquel cuyos elementos están regularmente separados, siendo muy utilizados por ejemplo lazos *for*, generación de bases de tiempo , índices de arreglos y escalas de gráficos. El operador que Matlab utiliza para generar este tipo de vectores es `(:)`. Veamos algunos ejemplos:

.Dados dos escalares *j* y *k* la operación :

`j:k` es lo mismo que generar un vector que siga la siguiente regla `[j,j+1,...,k]` .Por ejemplo `j=3, k=6`

`j:k` da como resultado el siguiente vector

```
ans =  
 3  4  5  6
```

si *j* es mayor que *k* da una matriz vacía como resultado.

.Dados los escalares *j, k, e i* la operación :

$j:i:k$ es lo mismo que generar el siguiente vector $[j, j+i, j+2i, \dots, k]$. Por ejemplo $j=3, i=2$ y $k=12$ da como resultado el siguiente vector

ans =

3 5 7 9 11

si ahora $j=0, k=1$ e $i=0.1$ la operación $j:i:k$ da como resultado el siguiente vector

ans=

0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000 1.0000

3.6. Concatenación de vectores y matrices.

Concatenación es el proceso de unir pequeños vectores o matrices para generar un vector o matriz mas grande. De hecho la generación de una matriz se logra concatenando elementos individuales. El operador de concatenación es el par de corchetes $[]$. Existen dos tipos de concatenación: horizontal, indicada por el operador $(,)$ y vertical indicada por el operador $(;)$. Por ejemplo:

Para vectores:

.dados los vectores $a=[1\ 2\ 3]$ y $b=[4\ 5\ 6]$ la concatenación horizontal da como resultado

$c=[a,b]$

$c =$

1 2 3 4 5 6

mientras que la concatenación vertical

$c=[a;b]$

$c =$

1 2 3

4 5 6

Para matrices:

.dadas la matrices $A=[1\ 2; 3\ 4]$ y $B=[5\ 6; 7\ 8]$

$C=[A,B]$ da como resultado

$C =$

1 2 5 6

3 4 7 8

y la concatenación vertical

$C=[A;B]$

$C =$

1 2

3 4

5 6

7 8

Nota: Tanto par la concatenación vertical como horizontal existen funciones que se pueden ejecutar desde la línea de comandos. Las mismas son:

.horzcat(A,B,C...)

.vertcat(A,B,C...)

3.7. Operaciones con filas y columnas

El operador $(:)$ es el utilizado para seleccionar filas, columnas y/o elementos de vectores, matrices y arreglos de dimensiones mayores.

Utilizaremos la matriz

$A =$

1 2 3

4 5 6

7 8 9

para ejemplificar las siguientes operaciones:

La operación $A(:,j)$ indica la j -ésima columna de A . Por ejemplo para $j=2$ en la matriz A :

```
ans =  
2  
5  
8
```

La operación $A(i,:)$ indica la i -ésima fila de A . Por ejemplo para $i=1$ en la matriz A :

```
ans =  
1 2 3
```

La operación $A(:,i)$ es lo mismo que rescribir A .

La operación $A(j:k)$ equivale a $A(j), A(j+1), \dots, A(k)$. Por ejemplo para la matriz A y $j=1$ y $k=2$:

```
ans =  
1 4
```

La operación $A(:,j:k)$ equivale a $A(:,j), A(:,j+1), \dots, A(:,k)$: Para A , $j=1$ y $k=2$:

```
ans =  
1 2  
4 5  
7 8
```

La operación $A(:, :, k)$ es para arreglos tri-dimensionales.

La operación $A(:)$ ordena todos los elementos de la matriz en una simple columna. Para A será:

```
ans =  
1  
4  
7  
2  
5  
8  
3  
6  
9
```

siguiendo la convención de las operaciones anteriormente descritas se pueden eliminar filas o columnas utilizando un par de corchetes vacíos. Veamos un ejemplo:

Si queremos eliminar la segunda columna de A , podemos escribir:

$A(:,2)=[]$ resultando en

```
A =  
1 3  
4 6  
7 9
```

Lo mismo para las filas, en ese caso la sintaxis es $A(2,:)=[]$, si se elimina un solo elemento de la matriz da como resultado un error ya que la matriz deja de ser una matriz.

3.8. Funciones básicas de Vectores y Matrices

A continuación enunciaremos algunas de las funciones relacionadas con las matrices:

Análisis matricial:

norm: Calcula la norma para una matriz o un vector

normest: Estima la norma de una matriz.

rank: Calcula el rango de una matriz.

det: Calcula el determinante de una matriz.

trace: suma los elementos de la diagonal.

orth: realiza la ortogonalización

Solución de ecuaciones lineales:

inv: Calcula la inversa de una matriz

cond: Calcula el número de condición para la inversión de una matriz.

chol: Realiza la factorización de Cholesky

lu: Realiza la factorización LU
qr: Realiza la descomposición ortogonal triangular de una matriz.
pinv: Calcula la Pseudoinversa.
 Funciones de matrices:
expm: Exponencial de una matriz
logm : Logaritmo de una matriz
sqrtm : Raíz cuadrada de una matriz.
funm: Evalúa la función general de la matriz.

3.9 Funciones de formato de salida

El comando *format* controla el formato numérico de los valores de salida, es decir como aparecen en la ventana de comandos. No influye en como Matlab calcula los resultados o como los graba.

Supongamos el siguiente vector $x = [4/3 \ 1.2345e-6]$ veremos como aparece en pantalla con los siguientes formatos:

```

.format short
  1.3333  0.0000
.format short e
  1.3333e+000  1.2345e-006
.format short g
  1.3333  1.2345e-006
.format long
  1.33333333333333  0.00000123450000
.format long e
  1.33333333333333e+000  1.234500000000000e-006
.format long g
  1.33333333333333      1.2345e-006
.format bank
  1.33      0.00
.format rat
  4/3      1/810045
.format hex
  3ff5555555555555  3eb4b6231abfd271
  
```

Además de los formatos mencionados también esta *format compact* que suprime la mayoría de las líneas en blanco que aparecen en la pantalla.

Para un mayor control de la salida se debe utilizar las funciones *sprintf* y *fprintf*.

3.10. Operaciones con cadenas de caracteres

Una 'cadena' es un arreglo de caracteres. cada caracter es representado internamente por su correspondiente código ASCII. Por ejemplo :

```
string='esto es una cadena de caracteres'
```

```
string =
esto es una cadena de caracteres
```

Para ver esa cadena en formato ASCII utilizamos la función *double* que provee Matlab. Por ejemplo `cadeascii=double(string)`

```

cadeascii =
Columns 1 through 13
 101 115 116 111 32 101 115 32 117 110 97 32 99
Columns 14 through 26
 197 100 101 110 97 32 100 101 32 99 97 114 97
  
```

```
Columns 27 through 32
 99 116 101 114 101 115
```

También se puede pasar un arreglo de caracteres ASCII al formato de cadena de caracteres. Siguiendo con el ejemplo anterior el siguiente comando

```
str=char(cadeascii)
```

da como resultado

```
str =
```

```
esto es una cadena de caracteres
```

Para comparar si dos cadenas de caracteres son iguales tenemos una función que se llama *strcmp* el resultado de esta evaluación será verdadera o falsa. Por ejemplo :

```
cade=strcmp(str,string)
```

```
cade =
```

```
1
```

También al igual que las matrices y vectores los arreglos de vectores se pueden concatenar, la función que realiza esta tarea es *strcat*. Por ejemplo:

```
strcat(str,string)
```

```
ans =
```

```
esto es una cadena de caracteresesto es una cadena de caracteres
```

Nota: La concatenación de caracteres no agrega espacios en blanco por lo tanto cuando tenga que usar esta función recuerde agregar espacios en blanco. Estos últimos también se pueden agregar por concatenación

Por ultimo destacaremos dos funciones útiles en la evaluación de caracteres. La función *isletter* comprueba si un caracter pertenece al alfabeto o es otro caracter ASCII. Por ejemplo

```
isletter('\')
```

```
ans =
```

```
0
```

```
isletter('n')
```

```
ans =
```

```
1
```

La otra función es *isspace* que entrega como resultado un vector con valor cero donde el caracter no es un espacio y valor uno cuando el caracter es un espacio. Por ejemplo:

```
isspace(str)
```

```
ans =
```

```
Columns 1 through 13
```

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

```
Columns 14 through 26
```

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

```
Columns 27 through 32
```

```
0 0 0 0 0 0
```

Ejercicios

1. Dados los vectores $a=[1\ 2\ 3]$ y $b=[1\ 2\ 3]'$ generar un vector c columna y un vector d fila concatenando a y b .
2. Dadas la matrices $A=[1\ 1\ ;\ 1\ 1]$ y $B=[2\ 2\ ;\ 2\ 2]$ generar por concatenación de las mismas la matriz $C=[1\ 1\ 2\ 2\ ;\ 1\ 1\ 2\ 2;\ 2\ 2\ 1\ 1;\ 2\ 2\ 1\ 1]$.
3. Dada las matriz $A=[1\ 2\ ;3\ 4]$ y utilizando el operador traspuesta y el de concatenación generar la matriz $B=[1\ 2\ 1\ 3;\ 3\ 4\ 2\ 4\ ;\ 1\ 3\ 1\ 2;\ 2\ 4\ 3\ 4]$.
4. Dado el vector $a=[1\ 2\ 3\ 4\ 5]$, utilizado la potencia de arreglos (array) y concatenación generar una matriz C de dimensiones 5×4 cuyas columnas sean los números del vector a elevados a la 1era, 2da, 3ra y 4ta potencia.
5. Dada la matriz $A=[1\ 2\ 3\ 4;\ 5\ 6\ 7\ 8;\ 9\ 10\ 11\ 12;\ 13\ 14\ 15\ 16]$ utilizando la notación $A(i,j)$ realizar la suma de los elementos de la diagonal principal. Verificar el resultado con la función *trace*.
6. Para la matriz del ejercicio anterior eliminar la fila 2 y 3 utilizando el operador $[]$ luego utilizando el concatenación y el operador $(:)$ volverlas a agregar.

4.Lenguaje de Programación en Matlab

4.1.Operadores de Relación

Las comparaciones en Matlab son realizadas con los operadores de relación, cuando la operación de comparación que se realiza es verdadera retorna un 1, cuando es falsa un 0. En el caso de trabajar con arreglos o matrices las comparaciones se realizan elemento a elemento, entregando como resultado una matriz o arreglo de las mismas dimensiones conformada por unos y ceros. Los operadores que utiliza Matlab son:

$>$, <i>mayor que</i> ,.	La expresión $A > B$ retornara 1 en aquellos elementos que satisfagan la expresión. Por ejemplo : $A=[1\ 2\ 3]$ $B=[3\ 2\ 1]$ retorna $ans = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$.
$<$, <i>menor que</i> , .	La expresión $A < B$ retorna $ans = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$
$>=$, <i>mayor o igual que</i> ,.	La evaluación de $A >= B$ retorna $ans = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$.
$<=$, <i>menor o igual que</i> ,.	La evaluación de $A <= B$ retorna $ans = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix}$.
$==$, <i>igual que</i> ,.	La evaluación de $A == B$ retorna $ans = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$.
\sim , <i>distinto de</i> ,.	La evaluación de $A \sim B$ retorna $ans = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$.

Par vectores y arreglos ambos comandos deben ser del mismo tamaño al menos que uno sea un escalar.

Por ejemplo utilizando el vector A es valida la expresión $A > 2$ la cual retorna $ans = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$

Es decir que cuando uno de los operados es un escalar y el otro no Matlab evalúa la expresión comparando el escalar con cada uno de los elementos del otro operando.

Además de los símbolos para realizar las operaciones de relación Matlab provee las siguientes funciones que pueden ser usadas desde la línea de comandos :

<i>eq</i>	- Equal	$==$
<i>ne</i>	- Not equal	\sim
<i>lt</i>	- Less than	$<$
<i>gt</i>	- Greater than	$>$
<i>le</i>	- Less than or equal	$<=$
<i>ge</i>	- Greater than or equal	$>=$

Por ejemplo $eq(A,B)$ es lo mismo que escribir $A == B$. Para obtener información de la sintaxis de esta funciones simplemente escriba *help* seguido del *nombre* de la función desde la línea de comandos.

4.2.Operadores Lógicos

Matlab tiene disponible tres operadores lógicos, &, | y ~. Cada operador lógico tiene un conjunto específico de reglas que determina el resultado de una expresión lógica.

Las operaciones que describen los símbolos anteriores son:

.& representa la operación AND, es decir el resultado es cierto si ambos operandos son lógicamente cierto. En términos numéricos un operando es cierto si es distinto de cero. Por ejemplo la operación lógica AND entre los dos vectores siguientes:

```
u = [1 0 2 3 0 5];
```

```
v = [5 6 1 0 0 7];
```

```
u & v
```

```
ans =
```

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

.| representa la operación lógica OR, es decir el resultado es lógicamente cierto si uno de los operandos es lógicamente cierto. El resultado será falso (0) si ambos operandos son cero. Por ejemplo la operación OR con los vectores anteriores dará como resultado:

```
u | v
```

```
ans =
```

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

~ representa la operación Not (negación), niega el operando. Produce un resultado cierto si el operando es falso y falso si el operando es cierto. Por ejemplo veamos la negación del vector u:

```
~u
```

```
ans =
```

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

Los operadores lógicos al igual que los de relación comparan los elementos correspondientes de arreglos con iguales dimensiones. Para vectores y arreglos ambos operandos deben ser del mismo tamaño al menos que uno de ellos sea un escalar. Para este caso Matlab evalúa el escalar comparándolos con cada uno de los elementos del arreglo. Donde la evaluación resulte verdadera el resultado será uno y donde sea falsa resultara cero.

Además de los símbolos para utilizar la operaciones lógicas Matlab provee las siguientes funciones que pueden ser ejecutadas desde la línea de comandos

and - Logical AND

or - Logical OR

not - Logical NOT

Por ejemplo la negación del vector u se puede escribir de la siguiente manera not(u) que es lo mismo que ~u.

Aparte de los tres operadores lógicos básicos Matlab tiene funciones lógicas específicas, como por ejemplo:

.xor. Esta función realiza la operación OR exclusiva entre los operandos, retorna verdadero (1) si uno de los operandos es verdadero y el otro es falso. En términos numéricos retorna 1 si uno de los operandos. Por ejemplo :

```
a = 1;
```

```
b = 1;
```

```
xor(a,b)
```

```
ans =
```

```
0
```

.all. Esta función retorna verdadero (1) si todos los elementos de un vector no son cero. En el caso de matrices opera sobre cada columna de la misma

Por ejemplo:

```
A = [0 1 2;3 5 0]
```

```
A =
```

```
0 1 2
```

```
3 5 0
```



```
all(A)
ans =
    0    1    0
```

.any . Retorna 1 si alguna elemento del argumento es verdadero o no cero. Como la función *all* opera sobre las columnas de una matriz.

Por ejemplo:

```
v = [5 0 8];
```

```
any(v)
ans =    1
```

Otras funciones lógicas de Matlab son *isnan*, *isinf* y *find*. La primera (*isnan*) retorna un 1 si encuentra un operando con valor NaNs, la segunda (*isinf*) retorna un 1 si el operando tiene valor Inf y por último la función *find* determina el índice del elemento del arreglo o matriz que cumple una condición lógica dada.

Es muy útil para crear mascarar y matrices de índices. En la forma mas general esta función retorna un vector de índices. Este vector puede ser utilizado para indexar arreglos de cualquier tamaño. Por ejemplo:

```
A = magic(4)
```

da como resultado la siguiente matriz

```
A =
    16    2    3   13
     5   11   10    8
     9    7    6   12
     4   14   15    1
```

```
i = find(A > 8);
```

```
A(i) = 100
```

```
A =
   100    2    3   100
     5   100  100    8
   100    7    6   100
     4   100  100    1
```

También se puede utilizar la función *find* para obtener índices de columnas y filas para arreglos rectangulares. Por ejemplo para la matriz anterior se podrían haber dos vectores que representen los índices de la matriz recorriendo la misma por fila o por columna , la sintaxis será :

```
[i,j]= find(A>8)
```

Ejemplos de uso de operadores

Veamos ahora un ejemplo utilizando operadores de relación y lógicos. Supongamos que tenemos una matriz con valores aleatorios y queremos seleccionar de ella los elementos que cumplan un determinado rango de valores para luego procesarlos de alguna manera.

Sea $A = \text{rand}(3,3)$ y queremos seleccionar aquellas entradas que satisfagan $A(i) \geq 0.75$ y $A(i) < 0.35$

```
A =
    0.4447    0.9218    0.4057
    0.6154    0.7382    0.9355
    0.7919    0.1763    0.9169
```

la búsqueda de los elementos se realiza de la siguiente manera

```
i=(A(i)>=0.75)|(A(i)<0.35)
```

donde i es un vector índice compuesto por ceros y unos, donde los unos indican los elementos de la matriz A que cumplen el requerimiento

```
i =
    0    1    0    1    0    1    0    1
```

luego para generar un vector con los datos que necesitamos utilizamos i de la siguiente manera

```
resul=A(i)
```

```
resul =
```

```
0.6154 0.9218 0.1763 0.9355
```

Esta última operación se conoce como el *direccionamiento lógico* .

4.3.Regla de Precedencia de operadores

Las expresiones que evalúa Matlab se pueden construir con cualquier combinación de operadores aritméticos, lógicos o de relación . Los niveles de precedencia determinan el orden en que Matlab evalúa una expresión. Dentro de cada nivel de precedencia los operadores tienen igual precedencia y son evaluados de izquierda a derecha. Las reglas de precedencia para los operadores de Matlab son mostradas en la siguiente lista y están ordenados desde el nivel mas alto al nivel mas bajo de precedencia.

1.Paréntesis ()

2.Traspuesta(.'), potencia(.^), traspuesta compleja conjugada(') y la potencia de matriz(^)

3.Incremento unitario(+), decremento unitario (-) y la negación lógica (~)

4. Multiplicación (.*), división derecha (./), división izquierda (./), multiplicación de matrices (*), división derecha de matrices(/) y la división izquierda de matrices(\)

5. Suma(+) y resta (-)

6. Operador dos puntos (:)

7.Menor que (<), menor o igual que (<=), mayor que (>), mayor o igual que (>=) , igual que (==) y no igual (~=).

8. la operación lógica AND (&)

9. La operación lógica OR (|)

Nota: El nivel predeterminado de precedencia puede ser cambiado utilizando paréntesis .

4.4. Sentencias de Control de Flujo

Matlab tiene ocho sentencias de control de flujo:

1) *if, else, and elseif:*

if evalúa una expresión lógica y ejecuta un grupo de sentencias basado en el valor del resultado de la expresión.

En su forma mas simple su sintaxis es:

```
if expresion_logica
```

```
    sentencias
```

```
end
```

Si la expresión lógica es verdadera (1), Matlab ejecuta todas las sentencias entre el *if* y la línea de *end*.

Si la evaluación de la expresión retorna falso(0) , Matlab saltea todas las sentencias entre el *if* y el *end* y continua la ejecución en la sentencia siguiente a la línea de *end*.

Por ejemplo:

```
if A(1,2) == 0
```

```
    disp('el elemento es cero')
```

```
    b=A(1,2);
```

```
end
```

Se pueden anidar cualquier numero de sentencias *if*.

Si la expresión lógica evalúa un valor no escalar todos los elementos del argumento deben ser distintos de cero. Es decir que si algún elemento de la matriz da como resultado falso las sentencias entre el *if* y el *end* no se ejecutaran. Esto es equivalente a realizar la siguiente verificación lógica:

```
if all(X(:))
```

```
    sentencias
```

```
end
```

Recordemos que la función lógica *all* retornaba un valor verdadero (1) si y solo si todos los elementos de

la matriz son distintos de cero.

Las sentencias *else* y *elseif* son sentencias que condicionan a posteriori la sentencia *if*.

.else .Esta sentencia no tiene condición lógica. Esta asociada con la ejecución de una sentencia *if* anterior o posiblemente una sentencia *elseif* si en estas mismas la evaluación de la expresión resulto ser falsa (0).

.elseif. Evalúa una condición lógica si la sentencia *if* o *elseif* precedente resulto ser falsa (0). las sentencias dentro de *elseif* se ejecutarán si la evaluación de la expresión asociada con este resulta verdadera (1). Se pueden anidar múltiples *elseif* dentro de un bloque *if*.

Por ejemplo:

```
if A(1)>=2
    A(1)=0
else
    A(1)=A(1)^2
end
```

La sentencia A(1)=0 se ejecuta si la expresión del if es verdadera y termina. La sentencia A(1)=A(1)^2 se ejecuta si la expresión del if resulta falsa.

```
if a>1
    a=b
elseif a>2
    a=c
elseif a>3
    a=d
else
    a=z
end
```

La sentencia a=z se ejecutara si y solo si las anteriores expresiones evaluadas en *if* y los *elseif*s son falsas.

2.Switch-case:

Switch ejecuta ciertas sentencias basado en el valor de una variable o expresión. Su forma básica es :

```
switch expresión(escalar o cadena)
    case valor1
        sentencias
    case valor2
        sentencias
    .
    .
    .
    otherwise
        sentencias
end
```

El funcionamiento este constructor es el siguiente: La palabra *switch* es seguida por una expresión o variable a ser evaluada. Luego puede seguir cualquier numero de grupos de sentencias *case*. Cada grupo consiste de la palabra *case* seguida de un valor con el cual se compara la expresión de la línea *switch*. Si la expresión es verdadera se ejecutan las sentencias luego del *case* y termina en *end* , caso contrario pasa a la línea del siguiente *case* si lo hubiera para su evaluación. Si ninguna de la evaluaciones *case* es verdadera todo el constructor *switch-case* no realiza nada. Al constructor se le puede agregar al final de todos los *case* la palabra clave *otherwise* y luego sentencias a ejecutar, estas sentencias se ejecutar si y solo si ninguna de las evaluaciones *case* es verdadera.

Switch trabaja comparando la expresión de entrada con cada valor case. Para expresiones numéricas una sentencia case es verdadera si (valor == expresión) es verdadera. Para el caso de cadenas de caracteres una sentencia case es verdadera si *strcmp*(valor ==expresión) es verdadera, *strcmp* es una función provista por Matlab que realiza la comparación de cadenas de caracteres.

El siguiente ejemplo muestra el funcionamiento del constructor switch-case:

```
x='perro'  
switch x  
case 'dog'  
    disp('el perro es ingles')  
case 'perro'  
    disp('el perro es español')  
  
otherwise  
    disp('de donde es?')  
end
```

Nota: si la expresión a evaluar en el case es una cadena de caracteres esta debe ir entre comillas para que el constructor la entienda como tal. ver el ejemplo anterior.

Nota:A diferencia de la programación en lenguaje C ,Matlab no ejecuta el siguiente case si el primero es verdadero. Por eso no utiliza las sentencias break.

A su vez puede evaluar múltiples condiciones en una simple línea encapsulando el case un arreglo. Veamos el siguiente ejemplo:

```
switch var  
    case 1  
        disp('1')  
    case {2,3,4}  
        disp('2 or 3 or 4')  
    case 5  
        disp('5')  
    otherwise  
        disp('something else')  
end
```

3.Lazos for

El lazo *for* ejecuta una sentencia o grupo de sentencias un predeterminado numero de veces.

Su sintaxis es

```
for index = start:increment:end  
    statements  
end
```

El incremento predeterminado es uno . Se puede especificar cualquier incremento , incluyendo incrementos negativos. Para índices positivos la ejecución finaliza cuando el valor del índice excede el valor final , para incrementos negativos la ejecución termina cuando el índice es menor que el valor final.

Por ejemplo este lazo se ejecuta cinco veces

```
for i = 2:6  
    x(i) = 2*x(i-1);  
end
```

Se pueden anidar múltiples lazos. Por ejemplo:

```
for i = 1:m
```

```

for j = 1:n
    A(i,j) = 1/(i + j - 1);
end
end

```

Nota: Cuando anidan lazos debe existir un *end* por cada *for*. La ejecución se puede acelerar reemplazando los lazos *for* con código vectorizado. Vectorizando el código de los archivos -m se logra mayor velocidad de procesamiento. donde otros lenguajes usan lazos For o Do, Matlab puede utilizar operaciones con vectores o matrices. Un simple ejemplo puede ser la creación de una tabla de logaritmos

```

x = .01;
for k = 1:1001
    y(k) = log10(x);
    x = x + .01;      % en esta línea radica la mayor diferencia de procesamiento
end

```

Mientras que la versión vectorizada del mismo código se vería de la siguiente manera

```

x = .01:.01:10;
y = log10(x);

```

Otro ejemplo sería la generación de una matriz cuyos elementos estén determinados por la ecuación $A(j,k) = \sin(j) \cdot \cos(k)$

Para generar esta matriz mediante lazos escribimos el siguiente código.

```

A=zeros(10) % genera una ,matriz vacia de 10 x 10

```

```

for j = 1:10
    for k = 1:10
        A(j,k) = sin(j) * cos(k)
    end
end

```

end

La versión vectorizada de este código sería:

```

k = 1:10;
A = sin(k)' * cos(K)

```

Como recomendación entonces, cuando la velocidad de procesamiento de datos es importante se deberían vectorizar los algoritmos para evitar lazos *for*.

Una propiedad muy importante de Matlab es que se pueden utilizar arreglos como índices de lazos *for*.

Por ejemplo si tenemos un arreglo A de m x n . La sentencia

```

for i = A
    sentencia
end

```

hace i igual al vector A(:,k) . Para la primera iteración del lazo k es igual a 1; para la segunda k es igual a 2 y así seguirá hasta que k iguale a n. Es decir que lazo itera un número de veces igual a la cantidad de columnas de A . Para cada iteración i es un vector conteniendo las columnas de A.

Por ejemplo el siguiente constructor de lazo

```

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

```

```

for i = A

```

```

    x = i

```

```

end

```

da como resultado

```

x =

```

```

    1

```

```

    4

```

```

    7

```

```
x =  
 2  
 5  
 8  
x =  
 3  
 6  
 9
```

4.Lazos while

El lazo *while* ejecuta una sentencia o grupos de sentencias repetidamente mientras la expresión de control sea verdadera. Su sintaxis es

```
while expression  
  statements  
end
```

Si la expresión evalúa una matriz todos los elementos de esta deben ser cero para que la ejecución continúe, un solo elemento de la misma que sea falso detiene la ejecución. Para convertir una matriz a un valor escalar que sirva de control utilizar las funciones *any* y *all*.

En general los lazos *while* son utilizados cuando el programador no conoce la cantidad de repeticiones a priori.

Por ejemplo necesitamos halla el numero entero n para el cual n! (n factorial) es un número de 100 dígitos

```
n = 1;  
while prod(1:n) < 1e100  
  n = n + 1;  
end
```

Se puede salir de un lazo *while* en cualquier momento utilizando la sentencia *break*.

5.Continue

La sentencia *continue* pasa el control a la próxima iteración de lazo *for* o *while* en el cual aparezca.

Salteando cualquier sentencia remanente en el cuerpo del lazo. Para el caso de lazos anidados pasa el control a la próxima iteración del lazo *for* o *while* al cual pertenece.

El siguiente ejemplo la sentencia *continue* pasa el control al lazo *while* si se cumple la expresión del *if*.

```
fid = fopen('magic.m','r');  
count = 0;  
while ~feof(fid)  
  line = fgetl(fid);  
  if isempty(line) | strcmp(line,'% ',1)  
    continue  
  end  
  count = count + 1;  
end  
disp(sprintf('%d lines',count));
```

6.Break

La sentencia *break* termina la ejecución de un lazo *for* o *while*. Cuando una sentencia *break* es encontrada, la ejecución continua con la próxima instrucción fuera del lazo. En el caso de lazos anidados la sentencia *break* sale del lazo mas interno y continua la ejecución del lazo que lo contiene

El siguiente ejemplo muestra un lazo *while* que lee el contenido de un archivo *fft.m* y lo pasa a un arreglo . La sentencia *break* es utilizada para salir del lazo *while* cuando la primer línea vacía sea encontrada.

```

fid = fopen('fft.m','r');
s = "";
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line)
        break
    end
    s = strvcat(s,line);
end
disp(s)

```

7. Try -catch

La forma general de una sentencia try-catch es de la siguiente manera:

```

try,
    statement,
    ...,
    statement,
catch,
    statement,
    ...,
    statement,
end

```

en esta secuencia las instrucciones entre la sentencia *try* y *catch* se ejecutan hasta que ocurre un error . Una vez que ocurre esto se ejecutan las sentencias que le siguen al *catch*. Matlab provee la función *lasterr* que le permite ver la causa del error . si un error ocurre entre la sentencia *catch* y *end* Matlab termina la ejecución .

8.Return

Return termina la ejecución en curso de comandos y devuelve el control la función que lo invoco o al teclado. Normalmente una función realiza sus tareas y devuelve el control una vez que alcanza la sentencia *end*. se puede intercalar en cualquier momento una sentencia *return* para devolver el control de quien llamo a la función

4.5. Programacion en Matlab

La programación en Matlab se realiza de dos formas básicas mediante los archivos M. Una forma de programación se llama *script* es la mas básica y consiste en generar una secuencia de comandos ,a ejecutar en forma secuencial, mediante un editor de texto y luego grabarla en formato .m .La otra forma se llama *function* o *m-function* y tambien es un secuencia de comandos generada con un editor de texto pero esta ultima acepta argumentos de entrada y devuelve resultados.

Entonces sabemos que hay dos tipos de archivos M el *script M-File* y el *Function M-File* .

script	function
No acepta argumentos de entrada o retorna argumentos de salida	Puede aceptar argumentos de entrada y retornar argumentos de salida
Opera sobre los datos del workspace	Las variables internas son locales a la función
Útil para realizar una serie de pasos de ejecución que se necesitan realizar repetidamente	Útil para extender y ampliar su aplicación mediante el lenguaje Matlab

4.5.1. Que es un archivo M?

En el siguiente código vemos las partes básicas de un archivo M tipo función de manera de

familiarizarnos con la programación en Matlab.

```
function f = fact(n) % Línea de definición de función.
% FACT Factorial. % Línea H1
% FACT(N) retorna el factorial of N, H! % Texto de ayuda
% con notacion N!
%Simplemente , FACT(N) es PROD(1:N).
f = prod(1:n); % Cuerpo de la función
```

Tomando esta ultima función como ejemplo veremos que tiene ciertos elementos comunes a todas la funciones en Matlab:

1ero. Una línea de definición de función. Esta línea define el nombre de la función y el numero y orden de argumentos de entrada y salida

2do. Una línea H1. significa línea de Help 1. Matlab muestra esta línea de la función cuando se ejecuta en la línea de comandos el comando *lookfor* o pide ayuda en un directorio completo.

3ero. Texto de ayuda. Matlab muestra el texto de ayuda junto con la línea H1 cuando ejecuta el comando *help* sobre la función

4to. El cuerpo de la función. Esta parte de la función contiene el código propiamente dicho que computa los valores y asigna los valores a los argumentos de salida

4.5.2. Script.

Es la forma mas simple de programación en Matlab ya que no tiene argumentos de entrada o de salida. Es útil para automatizar tareas repetitivas de comandos . Los *scripts* operan sobre datos existentes en el workspace o también pueden crear las variables sobre la cual operan. Cualquier variable que un *script* crea queda remanente en el workspace después que el *script* a finalizado de manera que pueden ser utilizados para posteriores procesamientos.

Veamos un ejemplo simple de un script:

Este calcula el rho para varias funciones trigonometricas y luego crea una serie de gráficos polares

```
% An M-file script to produce % Comment lines
% "flower petal" plots
theta = -pi:0.01:pi; % Computations
rho(1,:) = 2*sin(5*theta).^2;
rho(2,:) = cos(10*theta).^3;
rho(3,:) = sin(theta).^2;
rho(4,:) = 5*cos(3.5*theta).^3;
for i = 1:4
    polar(theta,rho(i,:)) % Graphics output
    pause
end
```

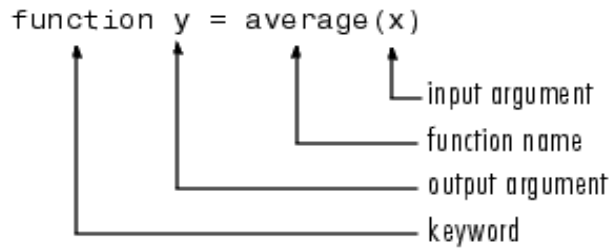
Para convertir este archivo de texto en un script de matlab lo debemos grabar con extensión .m , por ejemplo elegimos salvarlo como *petal.m* este archivo ahora es un script. Luego si desde la línea de comando ejecutamos el comando *petal* Matlab ejecutara las sentencias dentro del script.

Una vez que termina la ejecución las variables (i, theta y rho) permanecen en el workspace. Para ver estas ultima se abre la ventana del workspace o se escribe el comando *whos* en la línea de comando.

4.5.3.Función

Como ya dijimos la funciones se diferencian de los scripts en que esta pueden aceptar y entregar argumentos y opera sobre variables dentro de su propio workspace separado del workspace que se accede desde la línea de comandos. Veremos ahora en mas detalle como esta conformada una función. La línea de definición de función le informa a matlab que el archivo M abierto contiene una función y especifica la secuencia de llamada de los argumentos de la función.

Todas la funciones de Matlab tienen una línea de definición de función que sigue el siguiente patrón.



Si la función tiene múltiples valores de salida, se deben encerrar la lista de argumentos entre corchetes. Los argumentos de entrada deben ser encerrados entre paréntesis. Se deben utilizar comas para separar los argumentos. Veamos un ejemplo:

```
function [x,y,z] = sphere(theta,phi,rho)
```

Si no hubiera salida dejamos el argumento sin especificar o en blanco encerrándolo entre []. Por ejemplo veamos las dos siguientes líneas:

```
function printresults(x)
function [] = printresults(x)
```

Las variables que se pasan a la función no necesariamente tienen que tener el mismo nombre que en la definición de la función.

Cuando se ejecuta el comando lookfor y en nombre de la función matlab muestra la línea H1 de la función. Si ejecuta help aparecerá la línea H1 y todas las otras líneas de comentarios (precedidas del signo %) que estén entre la línea H1 y la primer línea de código. Las líneas de comentario que estén dentro del cuerpo de la función no serán mostradas por el comando help.

El cuerpo de la función contiene todo el código de la función Matlab. Puede consistir de llamadas a funciones, constructores, asignaciones, cálculos, comentarios y líneas en blanco.

Por ejemplo el siguiente es el cuerpo de una función que calcula el promedio de un vector.

```
[m,n] = size(x);
if ~(m == 1) | (n == 1) | (m == 1 & n == 1) % Flow control
    error('Input must be a vector') % Error message display
end
y = sum(x)/length(x); % Computation and assignment
```

Los nombres que se les pueden asignar a las funciones en Matlab tienen las mismas restricciones que los nombres de las variables. Deben comenzar con una letra y pueden seguir con todo tipo de símbolos.

El nombre del archivo de texto que contiene la función de Matlab debe ser el nombre de la función seguido de la extensión .m. Por ejemplo si la función anterior se llama average el archivo donde se guarda es average.m. Si el nombre del archivo y el de la definición de la función son diferentes matlab ignora este último y utiliza en la línea de comandos del nombre de archivo.

De todos modos aunque el nombre de la función y del archivo que la contiene no necesariamente tienen que ser el mismo se recomienda usar el mismo para ambos.

4.5.4. Como Trabaja una función en Matlab

Una función puede ser llamada desde la línea de comandos o desde adentro de otro archivo M. Hay que asegurarse de encerrar todos los argumentos en paréntesis y corchetes

Cuando Matlab llama a una nueva función este debe resolver que tipo de función es siguiendo los siguientes pasos:

Verifica si el nombre es un variable.

Verifica si el nombre es una subfunción, es decir una función Matlab que reside en el mismo archivo M de la función llamante

Verifica si el nombre es una función privada, es decir una función Matlab que reside en un directorio

privado, un directorio accesible solo accesible a archivos M que estén en el directorio por encima de este.

Verifica si el nombre de la función esta en la senda de búsqueda del Matlab. Matlab utiliza el primer archivo con el nombre especificado.

Si se duplican nombres de funciones Matlab ejecuta la primera que encuentra.

Cuando se llama una función ya sea desde la línea de comandos o desde otro archivo M, Matlab convierte la función en un pseudo código y lo mantiene en memoria hasta que se ejecute un comando de clear function o se salga de matlab. Esto evita tener que convertir una función cada vez que se llame.

Se puede utilizar la funcion clear de la siguiente manera para quitar funciones del workspace

Sintaxis

clear function_name	Quita una función específica de workspace
clear functions	Quita todas las funciones m compiladas
clear all	Quita todas las variables y funciones del workspaces

Desde la perspectiva del programador Matlab aparece como que pasa todos los argumentos de las funciones por valor. Sin embargo actualmente para hacer mas eficiente el uso de memoria Matlab solo pasa por valor aquellos argumentos que la función modifica. Si la funciona no modifica el argumento y solo lo usa para computación Matlab pasa el argumento por referencia

Workspaces de las funciones. Cada función tiene un área de memoria separada del workspace base de Matlab. Este área es llamada el función workspace con cada funcion teniendo su propio contexto de workspace

Las únicas variables a las que se puede acceder en Matlab son aquellas que se encuentren dentro del contexto que las llamo. Sin embargo se pueden definir a las variables como globales permitiendo que estas puedan ser obtenidas de otros workspaces.

Verificacion del numero de argumentos de una funcion. Las funciones *nargin* y *nargout* permiten determinar el numero de argumentos de entrada y salida respectivamente de una función. Entonces es muy útil para ejecutar programas que realizan diferentes tareas de acuerdo a la cantidad de argumento de entrada o salida de un función. Por ejemplo veamos la siguiente funcion:

```
function c = testarg1(a,b)
if (nargin == 1)
    c = a.^2;
elseif (nargin == 2)
    c = a + b;
end
```

Si el argumento de entrada es uno solo lo eleva al cuadrado, pero si el argumento es doble los suma

4.6. Variables locales y globales

Las mismas consideraciones que aplica a las variables en el workspace vale para la variables dentro de un archivo M:

.No se necesita declarar previamente las variables. Antes de asignar una variable a otra debe asegurarse que la que esta en lado derecho tiene un valor.

.Cualquier operación que asigne un valor a una variable la crea si esta no existe previamente. Si existe la sobrescribe .

.Los nombres deben iniciarse con una letra seguida de cualquier tipo de símbolo.

.Son sensibles a las letras mayúsculas

.Cada función tiene sus propias variables locales, que son diferentes de aquellas que están en el workspace base. Si se declara una variable como global esta puede ser utilizada por diferentes funciones.

Para que una aplicación Matlab funcione con variables globales se deben declarar todas las variables globales en todas las funciones que vayan a utilizarla . Para habilitar al workspace para que tenga

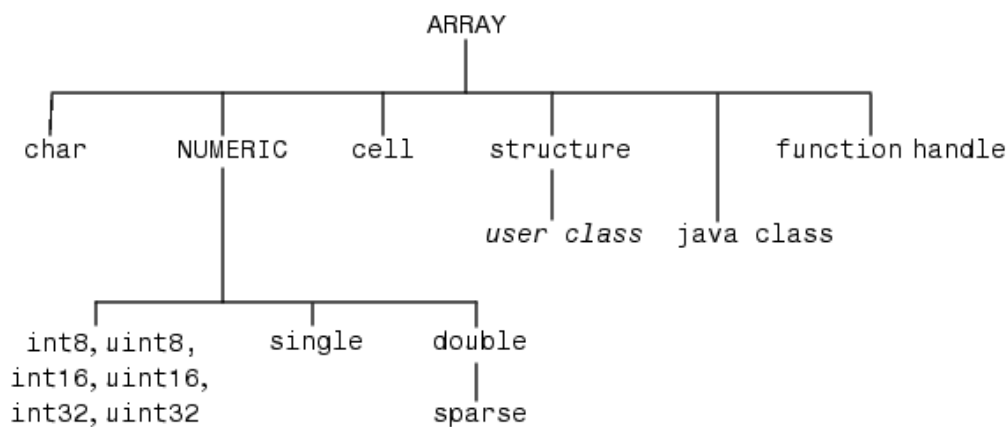
acceso a estas variables se deben declarar como globales en la línea de comandos
 Se recomienda declarar la variable global en el encabezado del programa, como en el lenguaje C ,
 aunque se puede inicializar antes de que sea usada en cualquier parte del programa.

Nota: Existen otro tipo de variables en Matlaba llamadas variables persistentes. Este tipo de variable no cambian de valor de una llamada de una funcion otra. Son utilizadas solamente entre funciones y se mantienen en memoria hasta que se ejecute un comando clear. Matlab provee funciones para evitar que las variables sen borradas del workspace, estas son *mlock*, *munlock* y *mislocked*.

4.7. Tipos de Datos en Matlab

En Matlab hay 14 tipos de datos . Cada uno de estos tipos de datos tiene la forma de un arreglo. el arreglo tiene la forma mínima de 0x0 y puede crecer hasta cualquier tamaño. Los arreglos de dos dimensiones son llamados generalmente matrices.

Todos los tipos de datos fundamentales son mostrados en letra minúscula en el siguiente diagrama.



A continuación describiremos los distintos tipos de datos :

.Arreglo de caracteres: En Matlab el termino string (cadena) se refiere a un arreglo de caracteres. Matlab representa internamente cada caracter como su correspondiente valor numérico. Se puede trabajar en ambas maneras, con el formato ascii y con la representación de caracter. Para crear este tipo de datos solamente tiene que ingresar los caracteres como quiera que aparezcan y encerrarlos entre comillas simples. Por ejemplo:

```
name = ' Thomas R. Lee ';
```

En el workspace la respuesta a un comando whos muestra

Name	Size	Bytes	Class
name	1x13	26	char array

Se puede ver que una cadena de caracteres le asigna 2 bytes por caracteres de almacenamiento interno. Para verificar si una función es de tipo char utilizamos la función *class* de la siguiente manera:

```
class(name)
ans =
char
```

Cuando se crean arreglos de caracteres bidimensionales, debemos asegurarnos que cada columna tenga el mismo largo. El siguiente ejemplo seria una entrada legal.

```
name = ['Thomas R. Lee' ; 'Sr. Developer']
```

```
name =  
Thomas R. Lee  
Sr. Developer
```

Cuando se crean arreglo bidimensionales de caracteres con diferentes largos se pueden rellenar los mas cortos con espacio en blanco para forzar a las filas tener el mismo largo.

Una forma simple de generar cadenas de caracteres es utilizando la función *char* , esta automáticamente realiza la complementación de las cadenas con espacios en blanco .Por ejemplo si nuestra cadena fuera:

```
name = ['Thomas R. Lee  '; 'Senior Developer'];
```

Para generar automáticamente dos cadenas del mismo largo escribimos la siguiente linea de comando:

```
name = char('Thomas R. Lee','Senior Developer')
```

Si ejecutamos la función whos veremos que ambas cadenas tienen el mismo largo

```
name      2x16      64 char array.
```

.arreglos de celdas de caracteres (strings). A menudo es conveniente guardar los grupos de cadenas en arreglos

de celdas en lugar de arreglos de caracteres estándar . Esto evita el engorroso trabajo de tener que convertir todas las cadenas al mismo largo . Evitando así un uso ineficiente de memoria . Matlab provee la función *cellstr* que convierte un arreglo de caracteres en un arreglo de celdas. Veamos el siguiente ejemplo:

```
data = ['Allison Jones';'Development  '; 'Phoenix  '];
```

Como podemos ver cada cadena está acomodada en largo con espacios en blanco, luego de ejecutar *cellstr* el resultado que obtenemos es el siguiente:

```
celldata = cellstr(data)
```

```
celldata =  
'Allison Jones'  
'Development'  
'Phoenix'
```

La función *iscellstr* determina si el argumento es un arreglo de celdas de caracteres. Retorna un uno en el caso de ser un arreglo de celdas

```
iscellstr(celldata)
```

```
ans =  
1
```

Si queremos verificar el largo de un elemento de arreglo de celdas utilizamos la misma función *length* que utilizamos para vectores. Por ejemplo :

```
length(celldata{3})
```

```
ans =  
7
```

.Arreglo de celdas. Un arreglo de celdas es un arreglo Matlab cuyas celdas son contenedores de datos que pueden soportar otro tipo de datos en Matlab. Por ejemplo un arreglo de celdas puede contener una matriz de números reales, un arreglo de caracteres y un vector de valores complejos. Veamos la siguiente grafica para clarificar los conceptos.

<p>cell 1,1</p> <table border="1"> <tr><td>3</td><td>4</td><td>2</td></tr> <tr><td>9</td><td>7</td><td>6</td></tr> <tr><td>8</td><td>5</td><td>1</td></tr> </table>	3	4	2	9	7	6	8	5	1	<p>cell 1,2</p> <table border="1"> <tr><td>'Anne Smith'</td></tr> <tr><td>'9/12/94'</td></tr> <tr><td>'Class II'</td></tr> <tr><td>'Obs. 1'</td></tr> <tr><td>'Obs. 2'</td></tr> </table>	'Anne Smith'	'9/12/94'	'Class II'	'Obs. 1'	'Obs. 2'	<p>cell 1,3</p> <table border="1"> <tr><td>.25+3i</td><td>8-16i</td></tr> <tr><td>34+5i</td><td>7+.92i</td></tr> </table>	.25+3i	8-16i	34+5i	7+.92i	
3	4	2																			
9	7	6																			
8	5	1																			
'Anne Smith'																					
'9/12/94'																					
'Class II'																					
'Obs. 1'																					
'Obs. 2'																					
.25+3i	8-16i																				
34+5i	7+.92i																				
<p>cell 2,1</p> <table border="1"> <tr><td>[1.43 2.98</td></tr> <tr><td>5.67]</td></tr> </table>	[1.43 2.98	5.67]	<p>cell 2,2</p> <table border="1"> <tr><td>7</td><td>2</td><td>14</td></tr> <tr><td>8</td><td>3</td><td>45</td></tr> <tr><td>52</td><td>16</td><td>3</td></tr> </table>	7	2	14	8	3	45	52	16	3	<p>cell 2,3</p> <table border="1"> <tr> <td data-bbox="933 787 1079 892">'text'</td> <td data-bbox="1096 787 1226 892"> <table border="1"> <tr><td>4</td><td>2</td></tr> <tr><td>1</td><td>5</td></tr> </table> </td> </tr> <tr> <td data-bbox="933 903 1079 997">[4 2 7]</td> <td data-bbox="1096 903 1226 997">.02 + 8i</td> </tr> </table>	'text'	<table border="1"> <tr><td>4</td><td>2</td></tr> <tr><td>1</td><td>5</td></tr> </table>	4	2	1	5	[4 2 7]	.02 + 8i
[1.43 2.98																					
5.67]																					
7	2	14																			
8	3	45																			
52	16	3																			
'text'	<table border="1"> <tr><td>4</td><td>2</td></tr> <tr><td>1</td><td>5</td></tr> </table>	4	2	1	5																
4	2																				
1	5																				
[4 2 7]	.02 + 8i																				

El ejemplo anterior se refiere a un arreglo de celdas de dos dimensiones. También estos pueden ser multidimensionales

Un arreglo de celdas se puede crear de dos formas diferentes:

- 1). Usando sentencias de asignación
- 2). Direccinando las celdas utilizando funciones especificas y cargando datos en las mismas.

1a) En el primer caso se le asignan datos a las celdas de dos formas diferentes el primero es el *indexado de*

celda. Se declara la celda a la que se le asignara el valor utilizando. Luego el valor es asignado encerrando el contenido que ira a la celda entre {}. Por ejemplo para crea un arreglo de 2x2

$A(1,1) = \{[1\ 4\ 3; 0\ 5\ 8; 7\ 2\ 9]\};$

$A(1,2) = \{'Anne\ Smith'\};$

$A(2,1) = \{3+7i\};$

$A(2,2) = \{-\pi; \pi/10; \pi\};$

La respuesta a un comando whos será:

whos A

Name	Size	Bytes	Class
A	2x2	644	cell array

Grand total is 45 elements using 644 bytes

1b).En el segundo caso se encierra el numero de celda entre {} y en el lado derecho de la igualdad se especifica el contenido de la celda en la forma normal en que el dato es utilizado en la ventana de comandos. Este segundo método se llama *indexado de contenido*. Veamos un ejemplo de la sintaxis utilizada

A{1,1} = [1 4 3; 0 5 8; 7 2 9];
 A{1,2} = 'Anne Smith';
 A{2,1} = 3+7i;
 A{2,2} = -pi:pi/10:pi;

La respuesta a un comando whos será la misma que para el caso anterior.

Si se asigna datos a una celda que esta fuera de las dimensiones originales del arreglo Matlab automáticamente expande para incluir la celda que el usuario especifica. Aquellas celdas adicionales que se agreguen como consecuencia de lo anterior quedaran vacías. Veamos un ejemplo con nuestro arreglo A de 2x2, si realizamos la siguiente asignación

A(3,3) = {5};

El arreglo de celdas pasa a tener la siguiente configuración que muestra la figura

cell 1,1 <table border="1" style="margin: auto;"> <tr><td>1</td><td>4</td><td>3</td></tr> <tr><td>0</td><td>5</td><td>8</td></tr> <tr><td>7</td><td>2</td><td>9</td></tr> </table>	1	4	3	0	5	8	7	2	9	cell 1,2 'Anne Smith'	cell 1,3 []
1	4	3									
0	5	8									
7	2	9									
cell 2,1 3+7i	cell 2,2 <table border="1" style="margin: auto;"> <tr><td>-3.14...3.14</td></tr> </table>	-3.14...3.14	cell 2,3 []								
-3.14...3.14											
cell 3,1 []	cell 3,2 []	cell 3,3 5									

Los {} son los constructores de celdas , tal como [] los son para los arreglos numéricos . Por lo tanto estos se comportan de modo similar a los anteriores , por lo tanto podemos también inicializar un arreglo de celdas de la siguiente manera

C = {[1 2], [3 4]; [5 6], [7 8]};

En este caso dentro de los{} se utilizan comas o espacios para separar por columna y el ; para separar por fila por lo tanto la declaración anterior daría como resultado el arreglo de la siguiente figura:

cell 1,1 [1 2]	cell 1,2 [3 4]
cell 2,1 [5 6]	cell 2,2 [7 8]

2) El segundo método para declarar un arreglo de celdas es utilizando la función *cell* .Esta función permite crear arreglos vacíos para luego ingresarle los valores. Por ejemplo :

`B = cell(2,3);`

Crea en arreglo vacío de dimensiones 2x2, luego mediante el método de asignación de celdas llenamos cada una individualmente.

`B(1,3) = {1:3};`

Se pueden obtener los datos del arreglo de celdas mediante dos métodos:

.Accediendo al contenido de las celdas usando *indexado de contenido*

.Accediendo a un subconjunto de celdas utilizando *indexado de celda*

En el primer caso consideremos por ejemplo el siguiente arreglo de celdas N de dimensiones 2x2

`N{1,1} = [1 2; 4 5];`

`N{1,2} = 'Name';`

`N{2,1} = 2-4i;`

`N{2,2} = 7;`

mediante este tipo de acceso especificamos la variable que va a recibir el contenido de la celda y en el lado derecho de la igualdad encerramos entre {} la celda de la cual queremos obtener el valor. Por ejemplo para el arreglo N

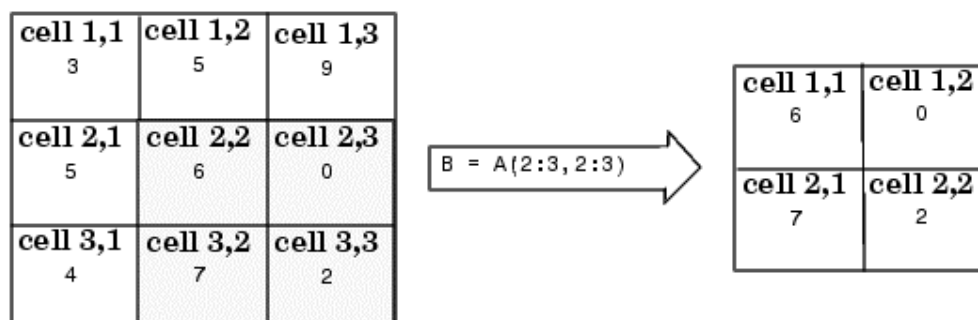
`c = N{1,2}`

`c =`

Name

esto indica que se esta asignando el contenido de la celda y no la celda en si misma.

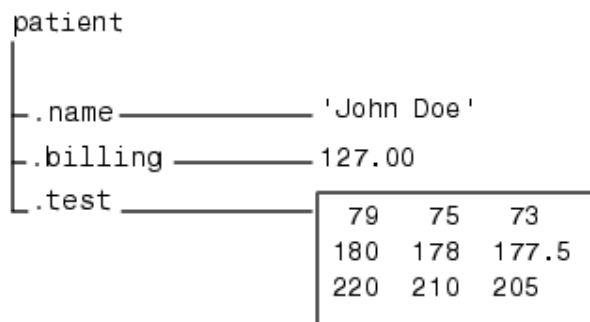
En el segundo caso , el indexado de celdas, se utiliza para asignar un conjunto de celdas a otras variables y de este modo crear un nuevo arreglo. Se debe utilizar el operador (:) para acceder a un subconjunto de celdas en un arreglo . Por ejemplo veamos la siguiente figura:



Para borrar celdas se utiliza el mismo mecanismo que para borrar columnas o filas de una matriz excepto que aquí estamos eliminando celda completas. entonces para borrar una celda se direcciona la misma y se le asigna el valor nulo mediante []. Por ejemplo veamos como es la sintaxis:

```
A(cell_subscripts) = []
```

.Estructuras. Las estructuras en Matlab son arreglos formados por contenedores de datos llamados campos. El campo de una estructura puede contener cualquier tipo de dato. Por ejemplo un campo puede contener un nombre, otro un numero y otro puede contener una matriz con los resultados de exámenes médicos. Las estructuras también pueden ser multidimensionales. Veamos la figura para aclarar este concepto de estructura



Los arreglos de estructuras se pueden construir de dos maneras diferentes:

- 1) Usando sentencias de asignación.
- 2) Usando la función struct

1) se puede construir un arreglo de estructuras de 1x1 asignando datos a los campos individuales. Matlab construye automáticamente la estructura a medida que se van ingresando los datos. Por ejemplo vamos a crear una estructura de 1x1 que represente los datos de un paciente

```
patient.name = 'John Doe';
patient.billing = 127.00;
patient.test = [79 75 73; 180 178 177.5; 220 210 205];
```

Ahora ingresando patient en la línea de comandos resulta en :

```
name: 'John Doe'
billing: 127
test: [3x3 double]
```

patient es un arreglo conteniendo una estructura con tres campos. Estos campos pueden tener datos de todo tipo. Para expandir el arreglo agregamos subíndice luego del nombre de la estructura. Veamos un ejemplo de como se ingresa

```
patient(2).name = 'Ann Lane';
patient(2).billing = 28.50;
patient(2).test = [68 70 68; 118 118 119; 172 170 169];
```

Ahora patient es un arreglo de estructuras de tamaño [1 2]. Una vez que un arreglo de estructuras

contiene mas de un elemento , Matlab no muestra el contenido individual de los campos cuando se escribe el nombre de la estructura en la línea de comandos .en su lugar aparece un sumario con información del arreglo de estructuras. Por ejemplo para nuestro caso escribir patient

```
patient
```

da como resultado

```
patient =  
1x2 struct array with fields:  
  name  
  billing  
  test
```

Se puede utilizar la función *fieldnames* para obtener información de los campos que conforman una estructura

A medida que el arreglo crece Matlab llena los campos que no se especifique con matrices vacías de manera que : Todas las estructuras dentro de un arreglo tienen la misma dimensión y todos los campos tienen el mismo nombre de campo.

Por ejemplo si ingresamos un nuevo paciente con los siguientes datos

```
patient(3).name = 'Alan Johnson'
```

es decir se ingresa solamente el nombre el arreglo se expande de la siguiente manera: queda con un tamaño [1 3] con los campos test y billing vacíos para el paciente 3

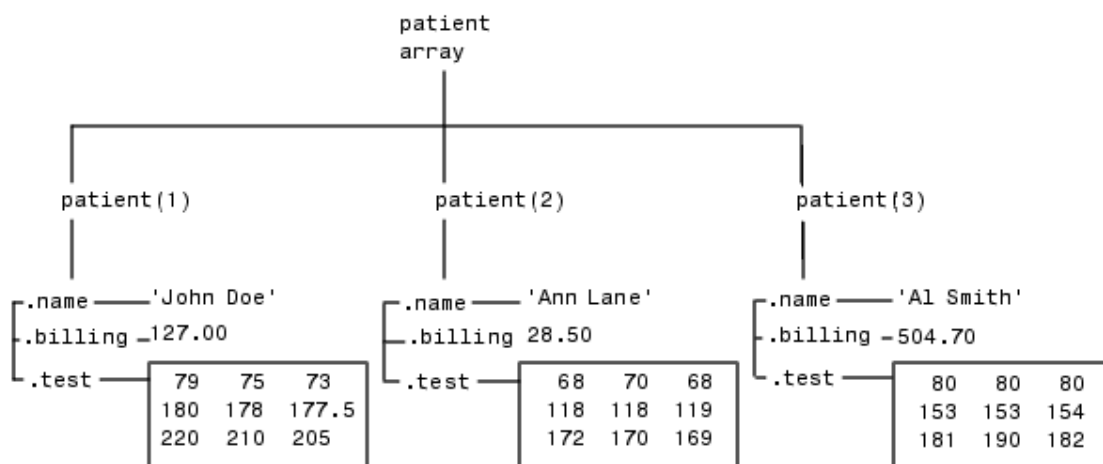
2) También se puede generar un arreglo de estructuras utilizando la función *struct* provista por Matlab. La sintaxis básica es

```
str_array = struct('field1',val1,'field2',val2, ...)
```

Donde 'field' indica el nombre que recibirá el campo y 'val' el correspondiente valor que se le asignará. Un campo puede tener un valor simple o puede ser cualquier tipo de dato soportado por Matlab , por ejemplo una matriz

Se pueden utilizar diferentes métodos para generar arreglos de estructuras. Estos difieren en la forma en que los campos de la estructura son inicializados.

Accediendo a los datos de un arreglo de estructuras. Usando direccionamiento de estructuras de arreglos se puede acceder el valor de un campo o un campo. Para ejemplo consideraremos la siguiente estructura de la figura



Podemos acceder a un subconjunto de arreglo utilizando subíndices. Por ejemplo creamos un arreglo llamado mypatients de modo que contenga los pacientes 1 y 2 del arreglo anterior

```
mypatients = patient(1:2)
```

1x2 struct array with fields:

```
name  
billing  
test
```

Entonces el primer paciente de mi estructura es el mismo que el arreglo original. Por lo tanto si ejecutamos

```
mypatients(1)
```

```
ans =  
    name: 'John Doe'  
    billing: 127  
    test: [3x3 double]
```

Para acceder a un campo en particular de esta estructura se debe escribir el nombre de la estructura seguido de un punto (.) y luego el nombre del campo que se quiere ver. Por ejemplo:

```
str = patient(2).name
```

```
str =  
Ann Lane
```

Para acceder a determinados elementos dentro de un campo utilizamos el siguiente indexado. Dado que el campo test es una matriz accederemos a un valor de la misma. Por ejemplo:

```
test2b = patient(3).test(2,2)
```

```
test2b =  
    153
```

El mismo método de indexado se utiliza para asignar valores a los campos de las estructuras

```
patient(3).test(2,2) = 7;
```

Otro aspecto interesante de estos arreglos en Matlab es que se pueden extraer los valores de un campo de varias estructuras. Por ejemplo la siguiente línea crea un vector de 1x3 conteniendo los campos billing de las tres estructuras que componen el arreglo.

```
bills = [patient.billing]
```

```
bills =  
    127.0000    28.5000    504.7000
```

También se puede crear un arreglo que contenga los campos test de los dos primeros pacientes. por

ejemplo:

```
tests = {patient(1:2).test}
```

```
tests =  
    [3x3 double]    [3x3 double]
```

Matlab provee las funciones *setfield* y *getfield* para el manejo de arreglos.

Nota: Se puede agregar un campo nuevo a la estructura en cualquier momento. Por ejemplo la siguiente instrucción agrega un nuevo campo a patient

```
patient(2).ssn = '000-00-0000';
```

Otra función de utilidad es *rmfield* que elimina un campo de una estructura. Por ejemplo la siguiente instrucción

```
patient = rmfield(patient,'name');
```

elimina el campo name del arreglo de estructuras patient.

Ejercicios

- 1) Escriba un función Matlab `[int,frac]=infr(x)` que tome un arreglo `x` de números no negativos y retorne los arreglos `int` y `frac` correspondientes a la parte entera y la parte fraccional respectivamente de todos los números de `x`.
- 2). Dado un arreglo `X` y un entero positivo `n` crear un arreglo `A` cuyos elementos sean los elementos de arreglo `X` repetidos `n` veces. Realizar una función `A=repli(X,n)` que realice esta operación.
- 3).Escriba una función `[nonz,media]=matriz(A)` que tome los valores distintos de cero de la matriz `a` y los entregue como resultado en un vector `nonz` y que calcule el valor medio de dicho vector y lo entregue en el escalar `media`.
- 4.)Escriba una función `dms(x)` que tome el numero `x` no negativo que representa un Angulo en grados y los traduzca y presente en pantalla en formato grados/min/seg
- 5). Escriba una función `t=allodd(A)` que tome un arreglo `A` y retorne 1 si todos sus valores son impares. Utilice la función *rem* para verificar la paridad.
- 6). Dados dos arreglos `A` y `B` no necesariamente de la misma dimensión escribir una función Matlab que genere un tercer arreglo intercalando los dos anteriores. Por ejemplo si `A=[1 2 3]` y `B=[1 2 3]` `C` sera `C=[1 1 2 2 3 3]`

5. Realización de Gráficos en Matlab

Matlab provee una variedad de funciones par mostrar datos de vectores en forma grafica así como también funciones para editar e imprimir esos gráficos. La siguiente tabla resume las funciones que producen las graficas básicas . Estas se diferencian en la forma en que escalan los ejes de la grafica. Cada una acepta entradas en la forma de matriz o vector y automáticamente escalan los ejes para acomodar los datos.

Función	Descripción
plot	Grafica de dos dimensiones con escalas lineales en ambos ejes
plot3	Grafica en tres dimensiones con escala lineal en ambos ejes
loglog	Grafica con escala logarítmica en ambos ejes
semilogx	Grafica con el eje x en escala logarítmica y eje y en escala lineal
semilogy	Grafica con el eje y en escala logarítmica y eje x en escala lineal
plotyy	Grafica con etiquetas en el lado derecho e izquierdo del eje y

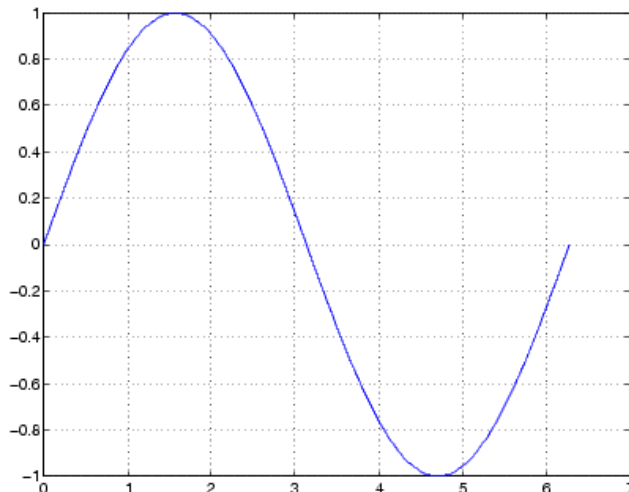
5.1.Como graficar una función.

Comenzaremos analizando el funcionamiento de la función plot.

La función plot trabaja de diferentes formas dependiendo de los argumentos de entrada. Por ejemplo si y es un vector $\text{plot}(y)$ produce un grafica lineal de los elementos de y versus los elementos de índice de y . Si se especifican dos vectores como argumentos ,por ejemplo $\text{plot}(x,y)$ produce el grafico de y versus x . Por ejemplo el siguiente enunciado crea un vector de rango $[0,2]$ en incrementos de $1/100$ y usa este vector para evaluar la función seno en ese rango. Matlab graficara el vector de rango en el eje x y el valor de la función seno en el eje y .

```
t = 0:pi/100:2*pi;  
y = sin(t);  
plot(t,y)  
grid on
```

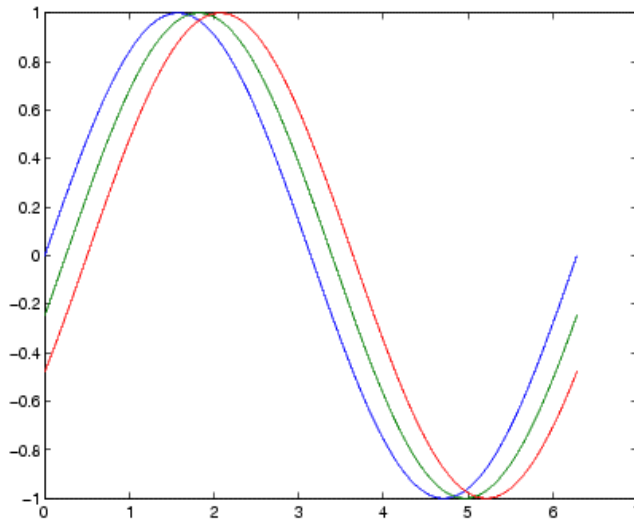
MATLAB automáticamente selecciona el rango de la grafica .



Se pueden graficar varias funciones en una sola llamada a la función plot. Matlab automáticamente cambia de color cada una de las graficas correspondientes a distintas funciones. Por ejemplo el siguiente enunciado grafica tres funciones seno de y versus tiempo t . Como en el ejemplo anterior el vector t es generado de la siguiente manera:

```
t = 0:pi/100:2*pi;  
y2 = sin(t-0.25);  
y3 = sin(t-0.5);  
plot(t,y,t,y2,t,y3)
```

El resultado será la siguiente grafica:

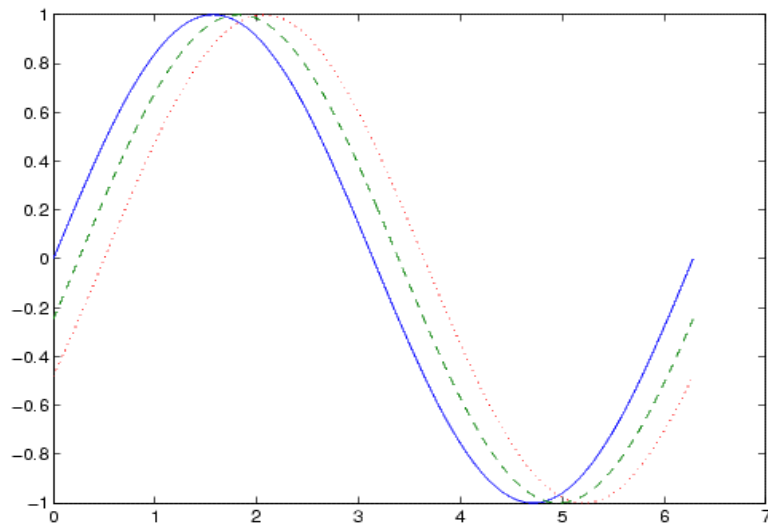


5.1.1. Estilo de líneas y marcadores

se puede asignar diferentes tipos de líneas a cada conjunto de datos. Esto se logra pasando el tipo de línea como argumento de la función plot. Por ejemplo, siguiendo con el ejemplo anterior:

```
t = 0:pi/100:2*pi;  
y = sin(t);  
y2 = sin(t-0.25);  
y3 = sin(t-0.5);  
plot(t,y,'-',t,y2,'--',t,y3,':')
```

En el argumento de plot se ven los diferentes identificadores de tipos de línea entre comillas simples. El grafico resultante será:



También es posible, además de especificar el estilo de línea, el color de las mismas y el tipo de marca para identificar el dato. La sintaxis es la siguiente:

```
plot(x,y,'color_style_marker')
```

color_style_marker es una cadena de caracteres que puede tener desde uno a cuatro caracteres encerrados en comillas simples identificando estos el color, el tipo de líneas y el tipo de marca. Las letras que identifican los colores son 'c', 'm', 'y', 'r', 'g', 'b', 'w', y 'k' que corresponde al cyan , magenta,yellow,red, green , blue, white y black.

Los caracteres utilizados para determinar los tipos estilos de líneas son: '-' para sólida, '--' para cortada, ':' para punteado, '-.' para cortado y punteado a la vez y 'none' sin línea.

Los marcadores son de dos clases: comunes '+', 'o', '*' y los tipo lleno , 's' para un cuadrado, 'd' para un diamante, '^' para un triangulo , 'v' un triangulo hacia abajo , '>' triangulo derecho, '<' triangulo izquierdo , 'p' una estrelle de cinco puntas , 'h' una estrelle de seis puntas y none para indicar ninguna marca .

Además del color y tipo de línea y marcador Matlab también permite controlar el ancho de la línea así como también el tamaño, color de fondo y color de borde del marcador.

La manera de especificar estas características es nuevamente a través del argumento de la función plot.

Las palabras claves (strings) que identifican las propiedades a modificar son las siguientes:

LineWidth - especifica el ancho de la línea en unidades de puntos

MarkerEdgeColor - especifica el color del marcador o el color del borde para marcadores llenos (círculo, cuadrado, diamante, pentagrama , hexagrama y los cuatro tipos de triángulos)

MarkerFaceColor - especifica el color de relleno para marcadores llenos.

MarkerSize - especifica el tamaño del marcador en unidades de punto

Por ejemplo:

```
x = -pi:pi/10:pi;
```

```
y = tan(sin(x)) - sin(tan(x));
```

```
plot(x,y,'--rs','LineWidth',2,...
```

```
    'MarkerEdgeColor','k',...
```

```
    'MarkerFaceColor','g',...
```

```
    'MarkerSize',10)
```

Produce un grafico con las siguientes características:

'--rs' indica línea cortada color roja con marcadores cuadrado

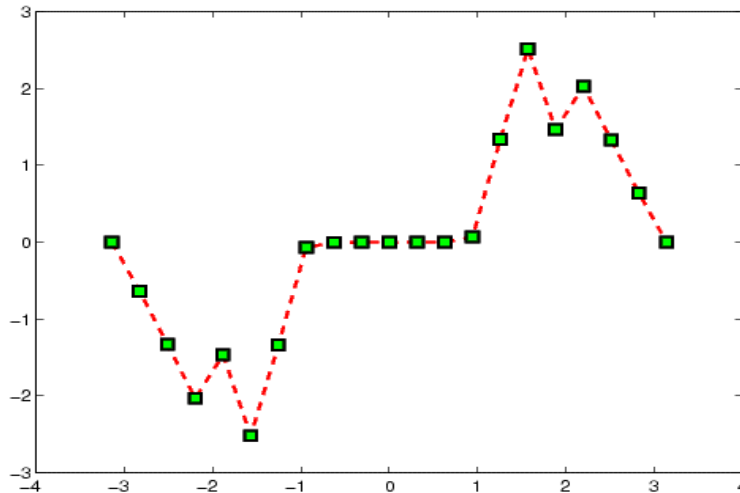
'LineWidth',2 indica una línea de ancho equivalente a dos puntos

'MarkerEdgeColor','k' indica el borde de los marcadores color negro

'MarkerFaceColor','g' el fondo de los marcadores es rellenado color verde

'MarkerSize',10 el tamaño de los marcadores es de 10 puntos

El grafico entonces quedaría de la siguiente manera:

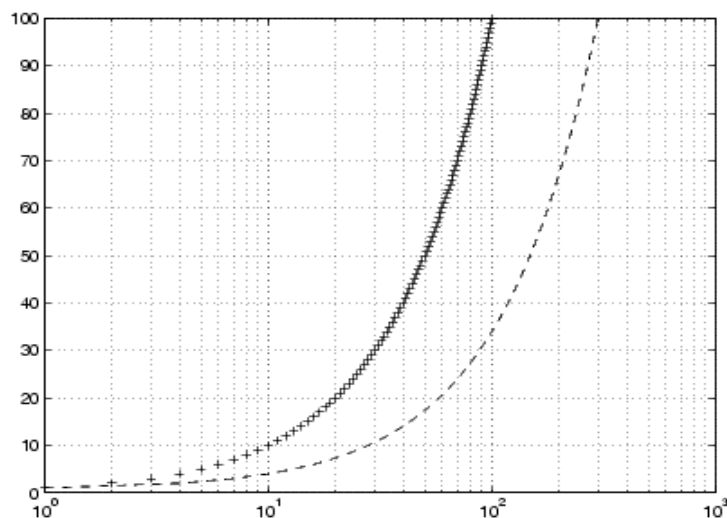


5.1.2. Agregar una grafica a una grafica existente

Una característica muy útil en las graficas de Matlab es que permite agregar una nueva grafica a una ya existente. Esto se logra utilizando la función *hold*. cuando se pone la función *hold on* Matlab no remueve ningún grafico existente sino que agrega los nuevos datos a la grafica ya existente, cambiando la escala automáticamente si es que los nuevos datos caen fuera del rango de la grafica anterior. Por ejemplo, las siguientes sentencias primero crea una grafica semilogarítmica y luego le agrega una lineal.

```
semilogx(1:100,'+')  
hold on  
plot(1:3:300,1:100,'--')  
hold off
```

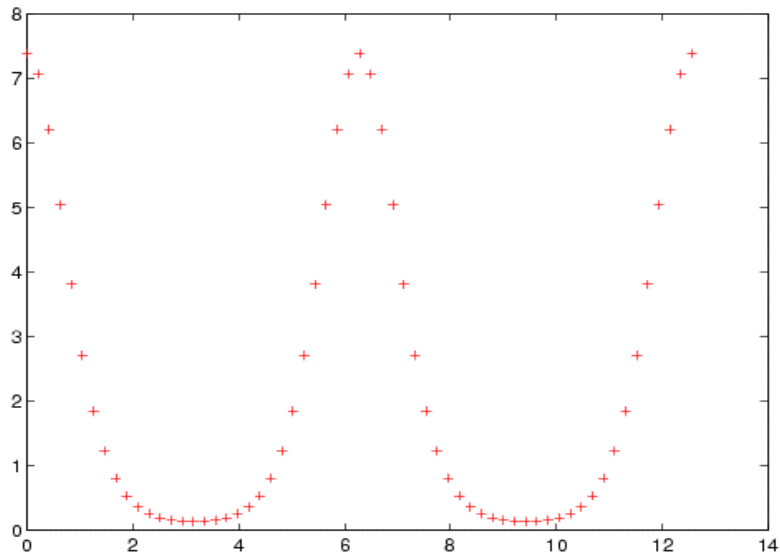
Aunque Matlab resetea la escala del eje x para acomodar los nuevos datos, no cambia la escala de logarítmica a lineal.



5.1.3. Graficar solamente los puntos de dato

En Matlab se pueden graficar los puntos de dato solamente. Esto se logra usando en el argumento de la función *plot* una especificación que no contenga estilo de linea. Por ejemplo el siguiente enunciado llama a la función *plot* y solo grafica las marcas

```
x = 0:pi/15:4*pi;
y = exp(2*cos(x));
plot(x,y,'r+')
```



También se pueden graficar marcas y las líneas continuas a la vez, el siguiente código realiza esto agregando la definición de línea continua en el argumento de la función plot.

```
x = 0:pi/15:4*pi;
y = exp(2*cos(x));
plot(x,y,'-r',x,y,'ok')
```

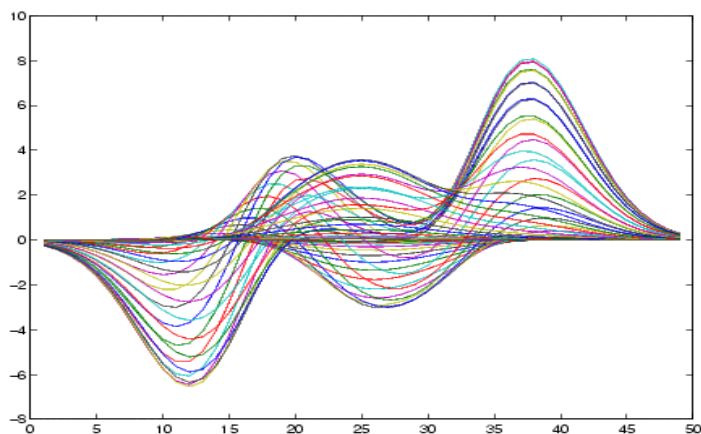
5.1.4. Grafica de datos de matrices

Cuando el argumento de la función plot es una matriz Matlab grafica una línea de grafico por cada columna de la matriz. El eje x estará etiquetado con el vector fila como índice, 1:m, donde m es el número de filas de la matriz.

Por ejemplo:

`Z = peaks;`

retorna una matriz de 49 x 49. La grafica de esta matriz, `plot(Z)` produce un grafico con 49 líneas distintas. Cada línea de color diferente es la grafica de una columna k versus los elementos de la fila.



La función `plot` también permite graficar una matriz versus un vector. Los argumentos entonces se ingresan de la siguiente manera , `plot(x,Y)` . Si los argumentos de la función son dos matrices grafica las columnas de una matriz versus las columnas de la otra.

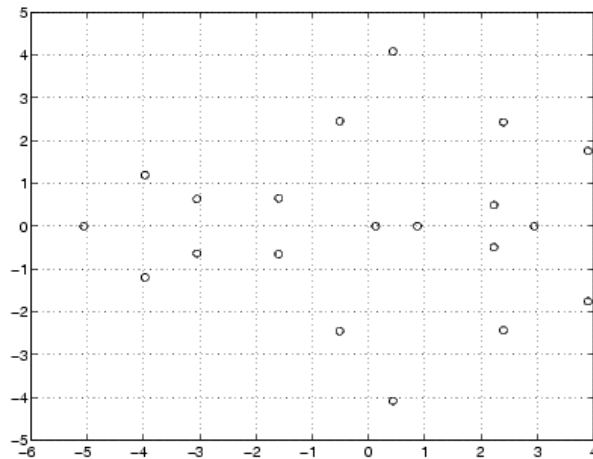
5.1.5.Grafica de datos imaginarios

Cuando el argumento de la función `plot` es complejo, es decir la parte imaginaria no es cero, Matlab ignora la parte imaginaria excepto que el argumento de `plot` sea es dato complejo en si mismo. En este ultimo caso la función grafica la parte real vs la imaginaria de la matriz o vector .Por lo tanto la sentencia `plot(Z)` es equivalente a

```
plot(real(Z),imag(Z))
```

Por ejemplo la siguiente sentencia grafica la distribucion de los autovalores de una matriz aleatoria usando marcadores circulares para indicar los puntos de datos.

```
plot(eig(randn(20,20)),'o','MarkerSize',6)
```



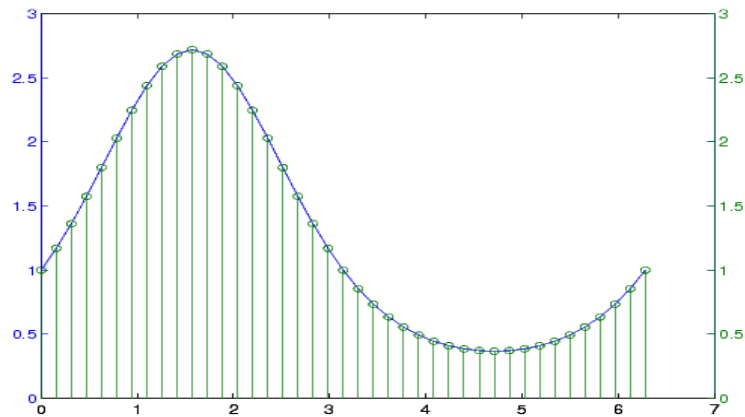
5.1.6.Grafica con dos eje Y

La función `plotyy` permite crear graficas con dos conjuntos de datos y permite utilizar el lado izquierdo y derecho del eje y para diferentes funciones

```
t = 0:pi/20:2*pi;  
y = exp(sin(t));  
plotyy(t,y,t,y,'plot','stem')
```

Esta función grafica la misma función en modo 'plot' y en modo 'stem' con las escalas de ambas graficas en la parte derecha e izquierda del eje y.

La grafica se vera de la siguiente manera:



Se puede utilizar la función `plotyy` para aplicar escalas logarítmica y lineal para comparar dos conjuntos de datos que tengan diferentes rangos de valores.

Por ejemplo:

```
t = 0:900; A = 1000; a = 0.005; b = 0.005;
```

```
z1 = A*exp(-a*t);
```

```
z2 = sin(b*t);
```

```
[haxes,hline1,hline2] = plotyy(t,z1,t,z2,'semilogy','plot');
```

Este ejemplo guarda los punteros de línea y ejes para luego modificarlos de la siguiente manera:

Primero: etiquetamos el eje y izquierdo de la siguiente manera

```
axes(haxes(1))
```

```
ylabel('Semilog Plot')
```

luego etiquetamos el segundo eje

```
axes(haxes(2))
```

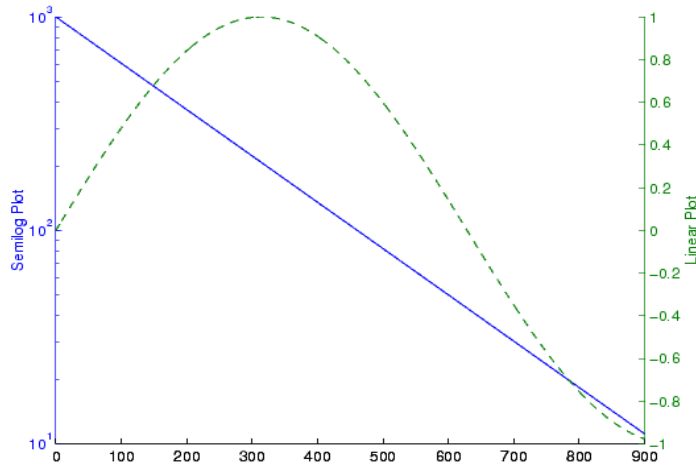
```
ylabel('Linear Plot')
```

En estas dos ultimas funciones la función `axes` opera sobre el eje que apunta el vector `haxes`

Luego de modo similar trabajamos sobre los estilos de línea, modificaremos para este ejemplo la grafica cuyo puntero a la grafica es `hline2`

```
set(hline2,'LineStyle','--')
```

La grafica resultante es la siguiente:



5.2. Presentación de la grafica (Adecuación de los ejes y presentación de escalas)

Cuando se crea un grafico , Matlab automáticamente selecta los limites de los ejes y el espaciamiento entre las marcas de acuerdo al conjunto de datos a graficar.

Sin embargo se pueden sobrescribir los valores predeterminados del grafico.

Esto puede ser realizado con los siguientes comandos:

axis - configura los valores de los ejes corrientes, se entiende por este ultimo al objeto recientemente creado o clickeado.

axes- crea nuevos ejes con características especificadas

get y *set* - permiten consultar y configurar una amplia variedad de propiedades de los ejes

gca -retorna el manejador (identificador) del eje corriente. si hay múltiples ejes en la figura los ejes corrientes son aquellos que se crearon ultimo o se clickearon recientemente.

Veamos como trabajan las funciones anteriormente mencionadas:

.*axis*: su forma mas simple es

axis([*xmin xmax ymin ymax*])

configura los limites del grafico con los valores que se pasan en el argumento [] de la función.

v=axis retorna un vector fila conteniendo los factores de escala de los ejes x ,y y z si lo hubiera.

v tiene cuatro o seis componentes dependiendo si es un grafico 2D o 3D

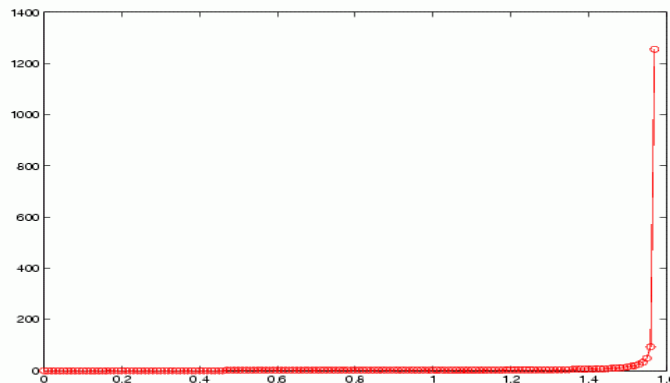
axes auto pone los ejes de la figura en el comportamiento predeterminado de Matlab, es decir que se calculan automáticamente los limites, sin embargo Matlab permite configurar auto cada eje por separado y así tener la libertad de configurar manualmente el resto de los ejes. Para ver todas las opciones buscar en index con la palabra clave '*axis*' y elegir *axis*[1] del resultado de la búsqueda.

Veamos el siguiente ejemplo para ver como trabaja *axis*. supongamos el siguiente enunciado

x = 0:.025:pi/2;

plot(x,tan(x),'-ro')

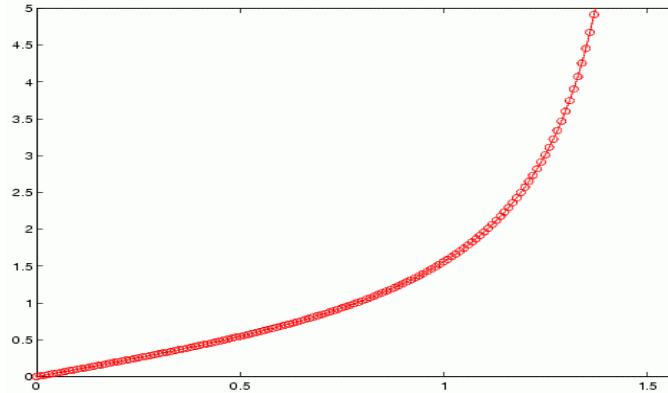
Con Matlab trabajando con la escala automática el grafico resultante es el siguiente:



Ahora si reconfiguramos las escalas con la función axis de la siguiente manera

```
axis([0 pi/2 0 5])
```

la grafica se vera:



.axes: es una función de bajo nivel para crear objetos gráficos 'ejes'
axes crea un nuevo objeto 'eje' en la figura corriente usando valores predeterminados
axes('PropertyName',PropertyValue,...) crea un objeto 'eje' con los valores especificados en la propiedad especificada en el argumento de la función. Aquella propiedad que no se especifique Matlab le asigna el valor predeterminado.

h = axes(...), retorna un manejador *h* del objeto 'eje' creado.

axes(h) convierte los ejes manejados por *h* en los ejes corrientes.

.get: esta función se utiliza para obtener las propiedades de un objeto.

get(h) retorna todas las propiedades y sus valores corrientes del objeto identificado por *h*

get(h,'Property name') retorna el valor de la propiedad pasada en el argumento del objeto apuntado por *h*.

a=get(h) retorna una estructura cuyos nombres de campo son los nombres de las propiedades del objeto apuntado por *h*. Si no fuera especificado el argumento de salida Matlab muestra los resultados en pantalla.

.set: configura las propiedades de un objeto.

set(H,'PropertyName',PropertyValue,...) configura la propiedad especificada en el argumento con el valor pasado en el argumento.

set(H,a) configura una propiedad determinada del objeto identificado por *H*. *a* es un arreglo de estructuras cuyos campos son los nombres de las propiedades del objeto.

set(H,pn,pv,...) configura las propiedades especificadas en el arreglo *pn* con los valores determinados por el arreglo *pv*

.gca: obtiene el puntero a los ejes corrientes.

h=gca retorna el puntero a los ejes corrientes de la figura corriente. Si no existieran los ejes Matlab los crea.

Vale aclarar que estas ultimas funciones trabajan en conjunto con *gcf* y *gco* Get Current Figure Get Current Object que son las encargadas de obtener el puntero al objeto deseado. Veamos el siguiente ejemplo para entender el uso de los punteros a los objetos y los manejos de las propiedades.

```
c=randn(1,10)
```

crea un vector aleatorio

```
plot(c)
```

grafica el vector

```
h=gcf
```

obtiene un puntero a la figura corriente

```
set(h,'Position',[0 0 200 500])
```

le asignamos a la figura una nueva posición y tamaño.

Las sentencias anteriores permiten acceder a las propiedades de la figura propiamente dicha. Ahora con el comando *gca* podemos acceder y modificar las propiedades de los ejes corrientes usando esta función del mismo modo que *gcf*. Siguiendo con el ejemplo anterior ejecutaremos las siguientes sentencias

```
h=gca                                obtenemos un puntero a los ejes corrientes
set(h,'xgrid','on','ygrid','on','linewidth',6)  modificamos las propiedades de los ejes
set(get(h,'XLabel'),'String','Values of X')     esta linea agrega texto a la etiqueta x
```

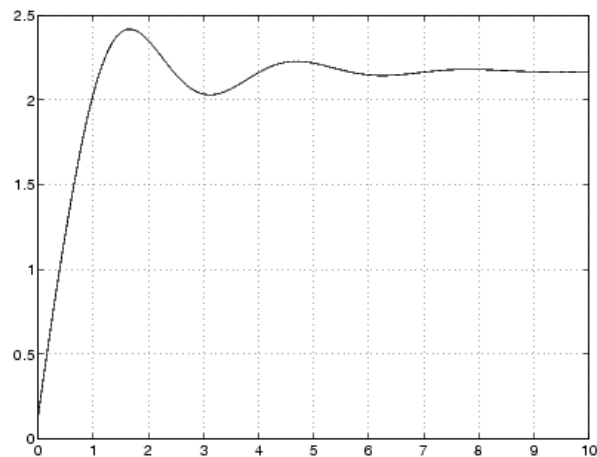
5.2.1.Limites y marcas en los ejes

Matlab selecta los limites de la grafica basado en el conjunto de datos a graficar. Sin embargo se pueden especificar los limites manualmente usando el comando *axis* como lo describieramos anteriormente.

```
axis([xmin,xmax,ymin,ymax])
```

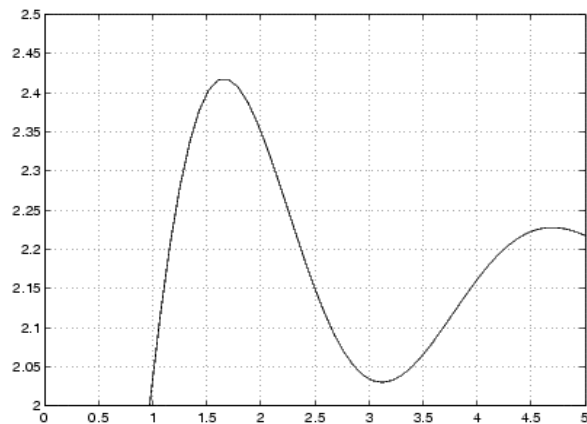
También se pueden seleccionar limites semiautomáticos de manera que podemos elegir los limites de un eje en particular dejando el o los otros en modo automático. Esto se logra ingresando el valor *Inf* en la función *axis*.

Por ejemplo el siguiente grafico usa la escala predeterminada de Matlab:



Supongamos ahora que limitamos los eje de la siguiente manera:

`axis([-Inf 5 2 2.5])`, la figura resultara:



Si ahora quisiéramos cambiar los valores ya sea de la escala x o y (2D) de manera de representar una distribución mas adecuada de los valores de la escala lo podemos realizar de la siguiente manera:

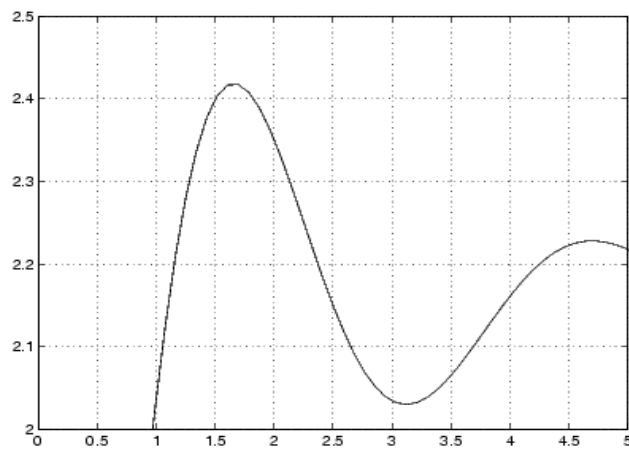
Se puede cambiar a traves de las propiedades de los ejes llamadas XTick e YTick. Esta define los valores de la escala ingresándolos como un vector de valores creciente e equidistantes.

Por ejemplo en la grafica anterior sin cambiar en la escala que estamos trabajando podemos cambiar la distribución de la escala de la siguiente forma:

```
set(gca,'ytick',[2 2.1 2.2 2.3 2.4 2.5])
```

{Nota: gca es la función que accede a la propiedad 'ytick' de los 'current axes'}

la grafica se reconfigurara de la siguiente manera

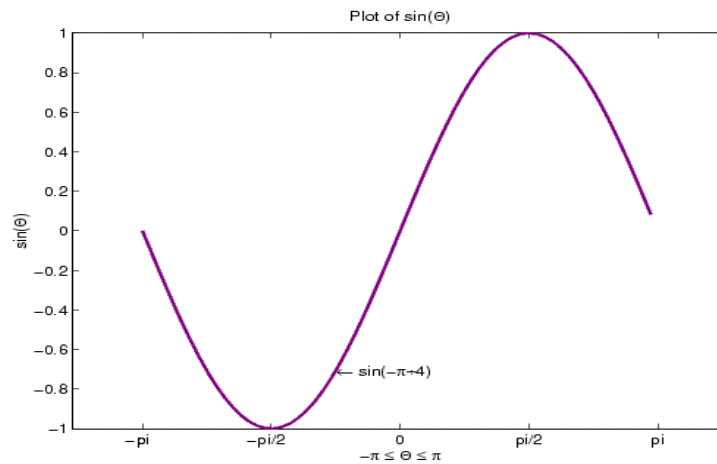


Ejemplo: La siguiente grafica muestra un ciclo de la función seno. Las siguientes sentencias nos permiten ingresar en la grafica los valores significativos en el eje x

```
x = -pi:1:pi;  
y = sin(x);  
plot(x,y)  
set(gca,'XTick',-pi:pi/2:pi)  
set(gca,'XTickLabel',{'-pi','-pi/2','0','pi/2','pi'})
```

Las siguientes funciones nos permiten ingresar una flecha que indica un valor determinado en la curva

```
text(-pi/4,sin(-pi/4),'\leftarrow sin(-\pi\div4)',...  
'HorizontalAlignment','left')
```



5.3. Ventanas de figuras (subgraficas)

Dentro de las opciones de grafica de Matlab esta la posibilidad de visualizar varias graficas en una misma figura. La función que permite esta operación es *subplot*. Esta divide la figura en subespacios. La sintaxis es la siguiente:

subplot(m,n,i) divide la figura en $m \times n$ secciones y grafica la figura corriente en la *i*esima sección. Por ejemplo las siguientes instrucciones grafican datos en cuatro diferentes regiones de la figura:

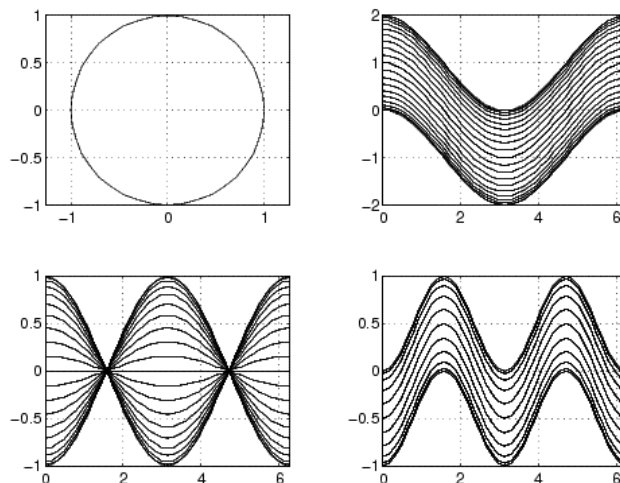
```
t = 0:pi/20:2*pi;
[x,y] = meshgrid(t);
subplot(2,2,1)
plot(sin(t),cos(t))
axis equal
```

se indica esto debido a la relación 3/4 de la pantalla de modo que grafique correctamente un círculo

```
subplot(2,2,2)
z = sin(x)+cos(y);
plot(t,z)
axis([0 2*pi -2 2])
```

```
subplot(2,2,3)
z = sin(x).*cos(y);
plot(t,z)
axis([0 2*pi -1 1])
```

```
subplot(2,2,4)
z = (sin(x).^2)-(cos(y).^2);
plot(t,z)
axis([0 2*pi -1 1])
```



Cada subregion contiene sus propios ejes con su propiedades y controles independientes de las otras subregiones en el ejemplo se puede ver como se utiliza la función *axis* para determinar los limites de de escala de cada subgrafica

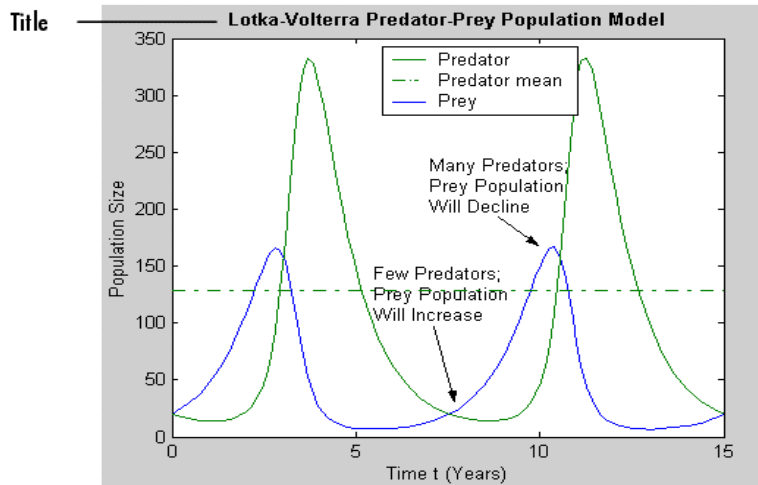
5.4.Formato de la grafica (títulos, etiquetas, textos ,legendas, flechas, etc.)

5.4.1.Titulo de la grafica

Es el texto que esta en la parte superior de la grafica. típicamente define el tema del grafico. Hay diferentes maneras de ingresarlo:

1) Usando la función *title* desde la línea de comandos. Este comando permite ingresar el texto del título en el mismo momento que se crea el gráfico.
 Por ejemplo para la siguiente figura

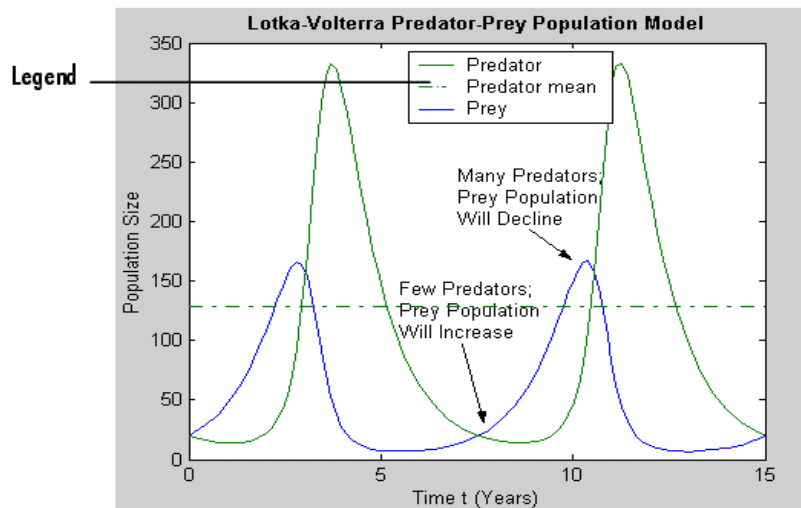
```
title('Lotka-Volterra Predator-Prey Population Model'),...
'FontWeight','bold')
```



- 2) Utilizando el editor de propiedades desde la ventana de la figura .
- 3) Utilizando el menu insert desde la ventana de la figura.

5.4.2. Legendas

Las legendas identifican cada conjunto de datos presentados en una gráfica. En la leyenda Matlab incluye una del tipo de línea o marcador que representa el dato. Matlab también incluye etiquetas de texto para identificar cada conjunto de datos.



La leyenda se puede ingresar desde:

- 1)El menú Insert en la ventana de la grafica.
- 2)Usando la función *legend* desde la línea de comandos. En los argumentos de la función se debe especificar el texto que acompañara la leyenda y a que conjunto de datos pertenece. Por ejemplo en la siguiente sentencia

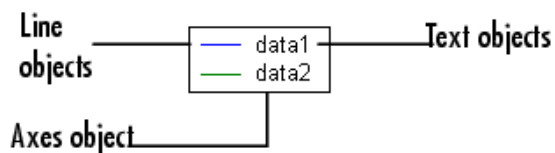
```
legend('Y1 Predator','Y2 Prey')
```

Predator es la leyenda que describe el conjunto de datos Y1 y Prey a Y2.

La función *legend* también nos permite determinar la otros aspectos como la posición.

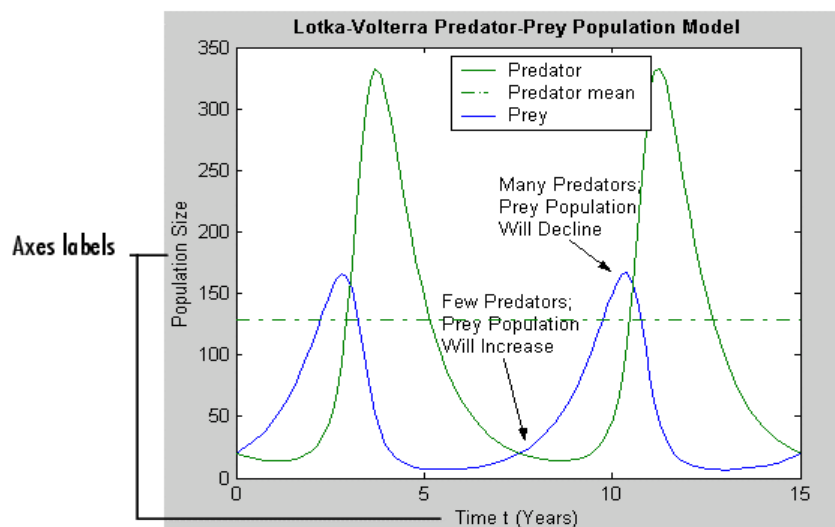
Para cambiar de posición la leyenda se hace click en la misma con el botón derecho del mouse cuando la figura esta abierta y se selecciona Unlock Axes Position . Luego manteniendo el botón izquierdo del mouse se arrastra la leyenda donde uno quiera.

La leyenda en la grafica es implementada como un objeto separado con sus propios ejes y texto. La siguiente figura representa los componentes de una leyenda:



5.4.3.Etiquetas de los ejes

Una etiqueta es un texto alineado con el eje al cual describe en la figura.



Las etiquetas se pueden ingresar de tres maneras diferentes:

- 1)Usando el menú Insert en la ventana de la grafica.
- 2)Usando el Editor de Propiedades del menú Edit en la ventana de la grafica.
- 3)Usando el comando *label* en la ventana de comandos. Se utilizan xlabel, ylabel y zlabel para etiquetar los respectivos ejes.

Por ejemplo veamos las siguientes líneas como ejemplo del comando label

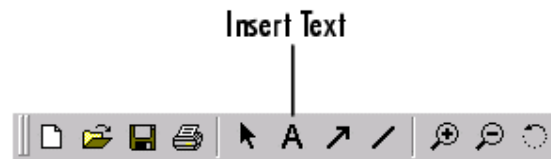
```
xlabel('t = 0 to 2\pi','FontSize',16)  
ylabel('sin(t)','FontSize',16)
```

Matlab posiciona automáticamente el texto de las etiquetas. Todo carácter que sea precedido de \ es interpretado como texto, refiriéndonos al ejemplo anterior \pi no coloca el valor numérico de pi sino su representación en caracter especial.

5.4.4. Anotaciones de Texto

Las anotaciones de texto se pueden agregar en cualquier parte de la figura. Hay dos formas de ingresar esta anotaciones en una grafica:

1)Utilizando la herramienta de agregar texto en la barra de tareas cuando la figura esta en modo edición. Se debe tener la precaución de configurar primero los ejes ya que el texto se adecua a la posición de los ejes y si un ajuste posterior de los mismos puede quedar mal presentado o fuera de la figura. El icono en la barra de tareas con el cual se accede a la herramienta de Texto es el de la figura

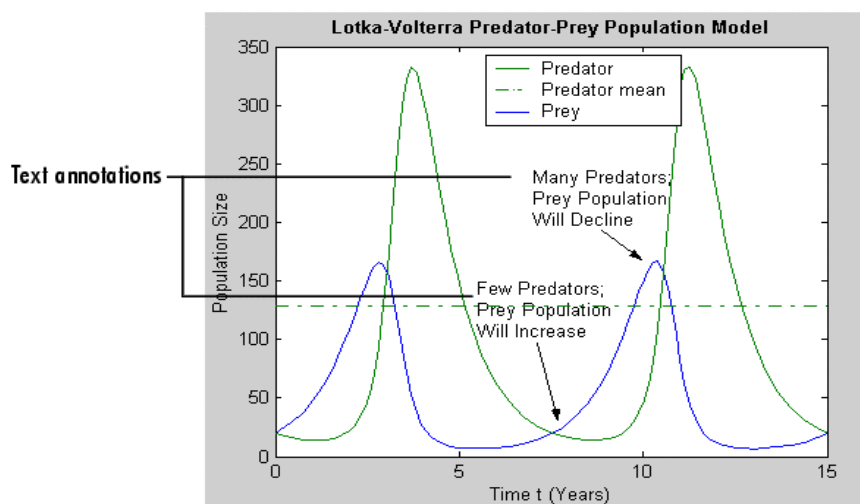


2)La otra forma de ingresar una anotación de texto es desde la línea de comandos con la función *text*. Esta función ingresa el texto y la posición del mismo en la grafica de acuerdo a las valores que se le hallan pasado en su argumento. Por ejemplo:

```
str1(1) = {'Many Predators;'};
str1(2) = {'Prey Population'};
str1(3) = {'Will Decline'};
text(7,220,str1)
```

```
str2(1) = {'Few Predators;'};
str2(2) = {'Prey Population'};
str2(3) = {'Will Increase'};
text(5.5,125,str2)
```

Agregan texto a la figura que venimos utilizando de ejemplo.



También se puede indicar que el texto se ubique en un valor específico de la grafica, (un valor de la

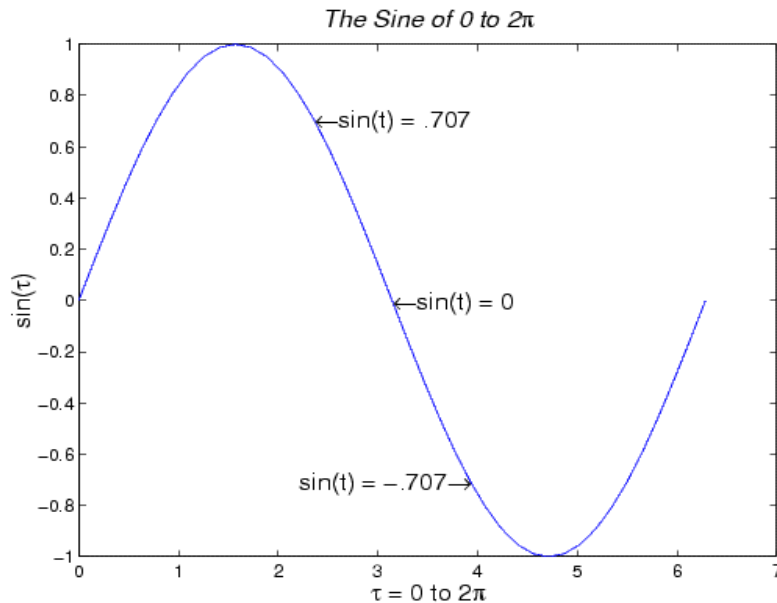
función graficada). Veamos un ejemplo con la función seno.

```
text(3*pi/4,sin(3*pi/4),...
      '\leftarrow sin(t) = .707',...
      'FontSize',16)
```

esta primera parte indica las coordenadas x e y del texto
se agrega una flecha izq. y el texto $\sin(t) = .707$
tamaño del texto

```
text(pi,sin(pi),'\leftarrow sin(t) = 0',...
      'FontSize',16)
text(5*pi/4,sin(5*pi/4),'sin(t) = -.707\rightarrow',...
      'HorizontalAlignment','right',...
      'FontSize',16)
```

Las siguientes líneas cumplen similares funciones, con lo cual la grafica quedara de la siguiente manera:



Una propiedad interesante de Matlab es que permite ingresar en el campo de texto. Por ejemplo cada fila del arreglo de caracteres Personaldata contiene información acerca de una persona.

```
PersonalData = ['Jack Straw ','489 Main St.','Wichita KN '];
```

Para poder mostrar estos datos como anotaciones de texto lo hacemos indexando el arreglo de la siguiente manera:

```
text(x1,y1,['Name: ',PersonalData(1,:)])
text(x2,y2,['Address: ',PersonalData(2,:)])
text(x3,y3,['City and State: ',PersonalData(3,:)])
```

Se puede también utilizar una variable numérica y mediante la función num2str convertirla a texto para luego agregarla en la anotación. Por ejemplo:

```
x = 21;
['Today is the ',num2str(x),'st day.']
Today is the 21st day.
```

concatenamos la cadena de caracteres
resultado de la concatenación

```
text(xcoord,ycoord,['Today is the ',num2str(x),'st day.'])
```

se utiliza la variable para hacer una anotación.

5.4.5. Flechas y Líneas

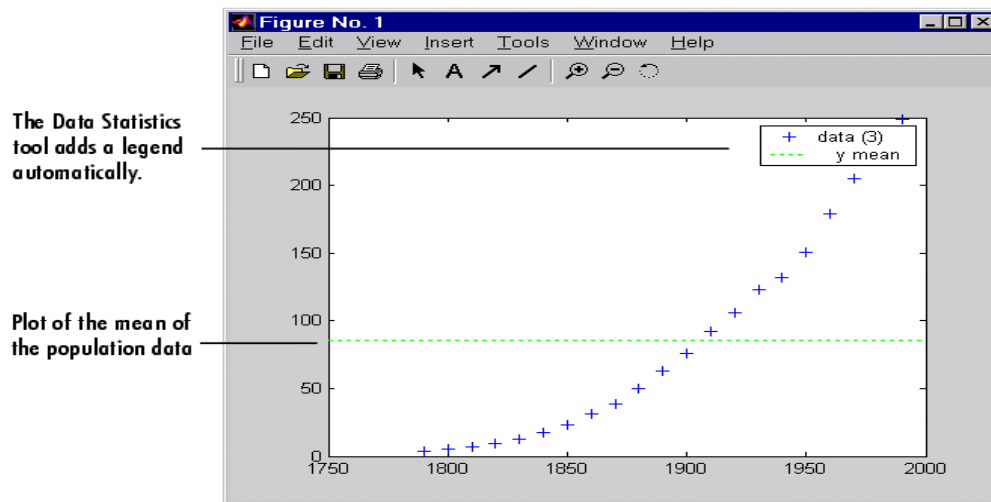
Se pueden agregar flechas y líneas a la grafica utilizando los siguientes iconos de la barra de tareas de la figura cuando la grafica esta en modo edición.



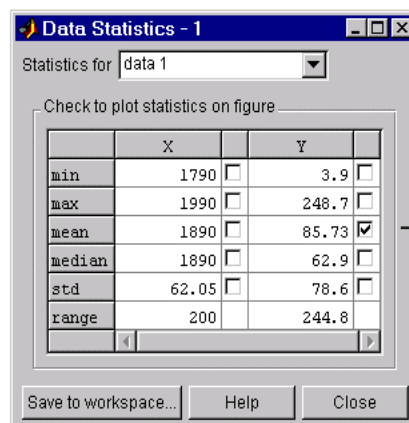
El agregado y edición de flechas y líneas es similar a la edición de anotaciones de texto, clikeando en cualquiera de estos iconos la grafica entra en modo edición automáticamente.

5.4.6. Estadística de la Grafica

Dentro del menú Tools (Herramientas) Matlab tiene un opción denominada Data Statistics, la misma es una herramienta que permite agregar la grafica de valores estadísticos (media, desvió estándar, etc) del dato seleccionado. Por ejemplo la siguiente grafica muestra los datos de crecimiento de la población en un periodo de tiempo determinado:



En color verde en esta grafica puede verse el valor medio de los datos representados por las cruces azules, este valor medio se agrego a la representación seleccionando las casilla del siguiente cuadro de dialogo. Note que se pueden seleccionar estos valores estadísticos con respecto a ambos ejes, y y x.



Select the statistic you want to plot by clicking in its check box.

5.5. Impresión y exportación

Estas son las opciones básicas de impresión:

1)Impresion.

.Impresion predeterminada: Se puede imprimir una figura a un archivo o a una impresora.

Si no se cambio la configuración predeterminada de Matlab se imprime una figura de 8 x 6 pulgadas sin el marco (frame) de la ventana que contiene la figura. Matlab selecciona el fondo de la figura en blanco y acomoda las marcas y las escalas al tamaño de la impresión tomando como referencia que se utiliza una hoja estándar de 8.5 x 11 pulgadas.

.Cambio de tamaño: Para imprimir una figura con el mismo tamaño que sale en pantalla, debemos seleccionar del menú File la opción Page Setup , una vez en este configuramos el tamaño de la hoja y la figura luego con el cuadro de dialogo Print imprimimos la figura.

.Impresión a un archivo: del menú file se selecciona Print esto invoca el cuadro de dialogo de impresión con la impresora predeterminada , en este cuadro de dialogo debemos tildar el cuadro de selección llamado Print to File . seleccionamos Ok y luego ingresamos en nombre del archivo y donde se guardara. Luego clickeamos OK.

Desde la línea de comandos podemos imprimir utilizando la función *print* por ejemplo:

```
print -f1-f2
```

imprime la figura 1 y la figura 2 hacia la impresora predeterminada.

2)Exportación.

La exportación de figuras se utiliza para grabar la figura en algún formato predeterminado para luego poder importarla desde algún otro programa, lo mas común para ser utilizadas en un editor de texto.

.Menu Export: Para exportar una figura a alguno de los formatos estándar de la industria seleccionamos el menú file de la figura y elegimos Export lo cual abre el cuadro de dialogo Export , luego usamos el botón Save para guardar grafico en el formato seleccionado en el cuadro Guardar Como.

Los formatos mas comunes de exportación son: BMP (bitmap), EMF(Enhanced Metafile), EPS(Postscript), TIFF, JPG, etc.

.Línea de Comandos : podemos exportar una figura con la función *print*, imprimiendo la grafica a un archivo utilizando el formato que le especifiquemos, o utilizando la función *save* en combinación con *cgf* (get Current Figure). Veamos los siguientes dos ejemplos realizan la misma tarea:

```
print -dmeta figurin
```

imprime la figura corriente con formato metafile (-dmeta) y con nombre de archivo figurin.

```
H=gcf;  
saveas(H,'figurin.emf');
```

obtiene un puntero a la figura corriente
graba la figura con nombre figurin y formato 'emf' metafile.

5.6.Graficas en 3D

Matlab provee varias funciones para graficar datos u objetos tridimensionales.

plot3 para graficar curvas en el espacio

mesh para graficar superficies mesh

surf para graficar superficies

contour para graficar contornos

También podemos nombrar *sphere* y *cylinder* que sirven para graficar superficies. Para obtener ayuda rapida escribimos en la linea de comandos *help graph3d*

5.6.1.Creación de graficas en 3D

La siguiente tabla muestra los pasos típicos a realizar para crear una grafica en 3D.

Paso	Código Tipico
generación de los datos	<code>Z = peaks(20);</code>
selección de la región de la figura donde se graficara	<code>figure(1) subplot(2,1,2)</code>
llamada a la función graficadora en 3D	<code>h = surf(Z);</code>
configurar el mapa de colores y el algoritmo de sombreado	<code>colormap hot shading interp set(h,'EdgeColor','k')</code>
agregar iluminacion	<code>light('Position',[-2,2,20]) lighting phong material([0.4,0.6,0.5,30]) set(h,'FaceColor',[0.7 0.7 0],... 'BackFaceLighting','lit')</code>
configurar el punto de vista	<code>view([30,25]) set(gca,'CameraViewAngleMode','Manual')</code>
configurar limites de los ejes y marcas	<code>axis([5 15 5 15 -8 8]) set(gca,'ZTickLabel','Negative Positive')</code>
configurar la relación de aspecto	<code>set(gca,'PlotBoxAspectRatio',[2.5 2.5 1])</code>
agregar etiquetas , legendas y texto	<code>xlabel('X Axis') ylabel('Y Axis') zlabel('Function Value') title('Peaks')</code>
impresion	<code>set(gcf,'PaperPositionMode','auto') print -dps2</code>

La contrapartida de la función plot en 3D es plot3 . Plot3 grafica un linea en 3D cuyas cordenadas estan dadas por los puntos de los vectores x, y, y z que a su vez deben ser del mismo largo.

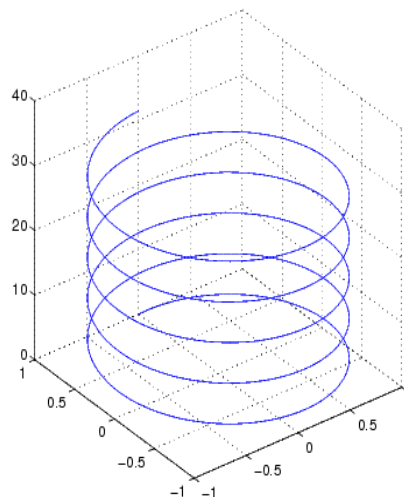
La sintaxis es:

`plot3(x,y,z)`

Por ejemplo:

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square; grid on
```

resulta en la siguiente grafica



Si los argumentos de la función plot son matrices las líneas de plot se realizan evaluando columna a columna, es decir la columna 1 de X contra la columna 1 de Y y Z y así sucesivamente se arma la grafica.

Por ejemplo:

```
[X,Y] = meshgrid([-2:0.1:2]);
```

genera dos matrices grilla Y y X

```
Z = X.*exp(-X.^2-Y.^2);
```

genera una matriz Z en funcion de X e Y

```
plot3(X,Y,Z)
```

grafica el resultado

```
grid on
```

la grafica resultante es

