Curso de Matlab

Entorno interactivo de cálculo y visualización vinculado a un lenguaje de programación de alto nivel

Dr. Sergio Giner

Profesor Adjunto DE

Cátedra de Simulación de Procesos I

Área Departamental Ingeniería Química

Facultad de Ingeniería-UNLP (2008)

CAPITULO 1. DIAGRAMACION

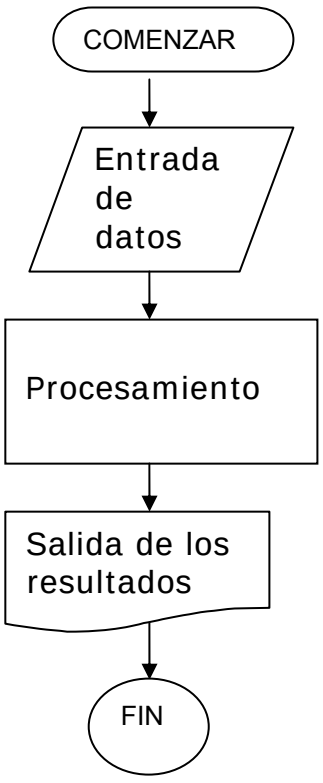
Introducción

Para resolver problemas de ingeniería, así como de otras disciplinas, siempre es necesario contar con los datos suficientes, y con un método donde procesarlos para hallar lo que se busca, los resultados. Cuando esta práctica se implementa mediante programación en una computadora, resulta importante bosquejar, esquematizar, diagramar el método de resolución en forma gráfica, antes de pasar a la programación propiamente dicha, o codificación en lenguaje de alto nivel. Precisamente la *Diagramación en Bloques*, o más genéricamente, la *Diagramación*, se define como "la representación gráfica del método de resolución de un problema, empleando símbolos normalizados". En esta clase, se verán ejemplos de *Diagramas de Bloques*, que representen las estructuras de diagramación típicas que pueden aparecer en casi todos los problemas. En algunas bibliografías los diagramas se denominan "diagramas de flujo" o, en Inglés "Flow charts". En esta cátedra, se va a hacer una diferenciación entre los Diagramas de Bloques, que, correspondiendo a las soluciones a problemas pertenecen al mundo abstracto de los programas, o *"Software"* y los *Diagramas de Flujo*, que estrictamente describen el funcionamiento de dispositivos físicos, tales como circuitos electrónicos, o la organización de los componentes de una computadora, y que por tanto pertenecen al mundo concreto del *"Hardware".*

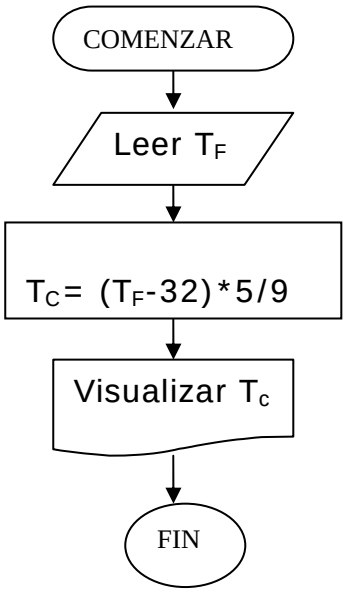
Volviendo al Diagrama de Bloques, éstos, en general proveen una organización lógica previa, que es útil independientemente del lenguaje de programación en el que se implemente finalmente la resolución (distintas versiones de C, Pascal, Visual Basic, Fortran o, como en este curso, Matlab).

Diagrama de Bloques generalizado

Podríamos decir que todos los problemas que pueden implementarse y resolverse en la computadora, responden a este diagrama de Bloques general:



Por ejemplo, la aplicación de este concepto al caso simple de la conversión de una Temperatura en º F a ºC, se exhibe a continuación:



Este diagrama corresponde al caso más sencillo de la estructura a su vez más simple, la "secuencial". Se define una estructura secuencial, como "aquella en la que las instrucciones se ejecutan una después de la otra, en el mismo orden en que están escritas"

Estructuras en Diagramación

En general, la solución de un problema típico, esto es, un diagrama de bloques (DDB) usual, combina las tres estructuras básicas que existen en diagramación.

a) Estructura secuencial

b) Estructura alternativa

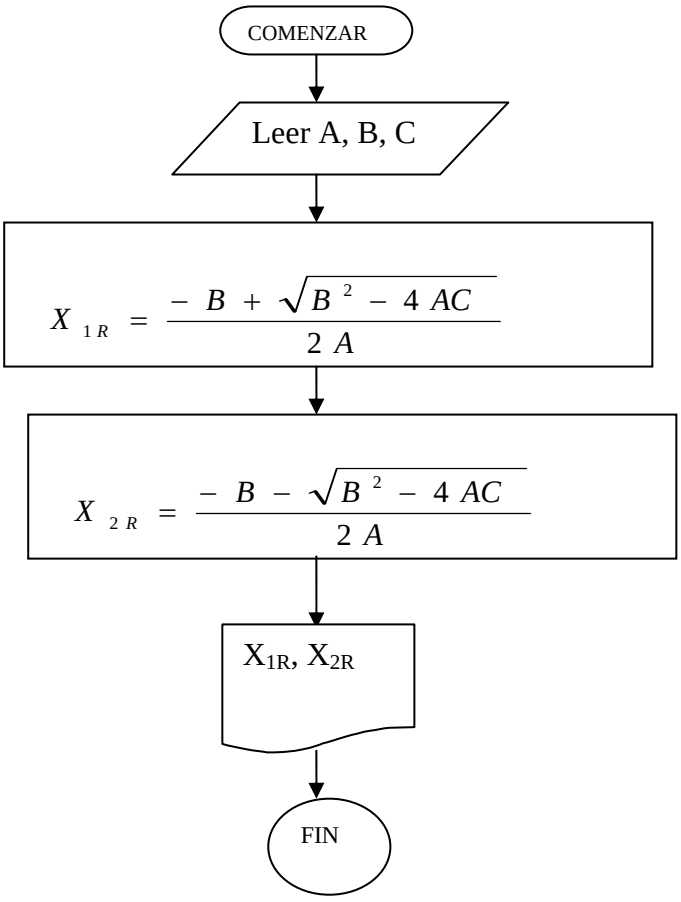
c) Estructura repetitiva

Apelaremos a la estructura secuencial pura, cuando tengamos una secuencia simple de instrucciones, donde el control de la ejecución del programa pase sólo una vez por cada instrucción. Tendremos una estructura alternativa cuando debamos tomar decisiones sobre el camino a seguir entre dos o más posibilidades, luego de cumplida una cierta instrucción. A su vez, llegaremos a la estructura repetitiva, cuando debamos recorrer muchas veces el conjunto de instrucciones, para distintos valores de determinadas variables.

Resolviendo la ecuación cuadrática $A X^2 + B X + C = 0$, podremos dar ejemplos de DDB que incluyan cada tipo de estructura.

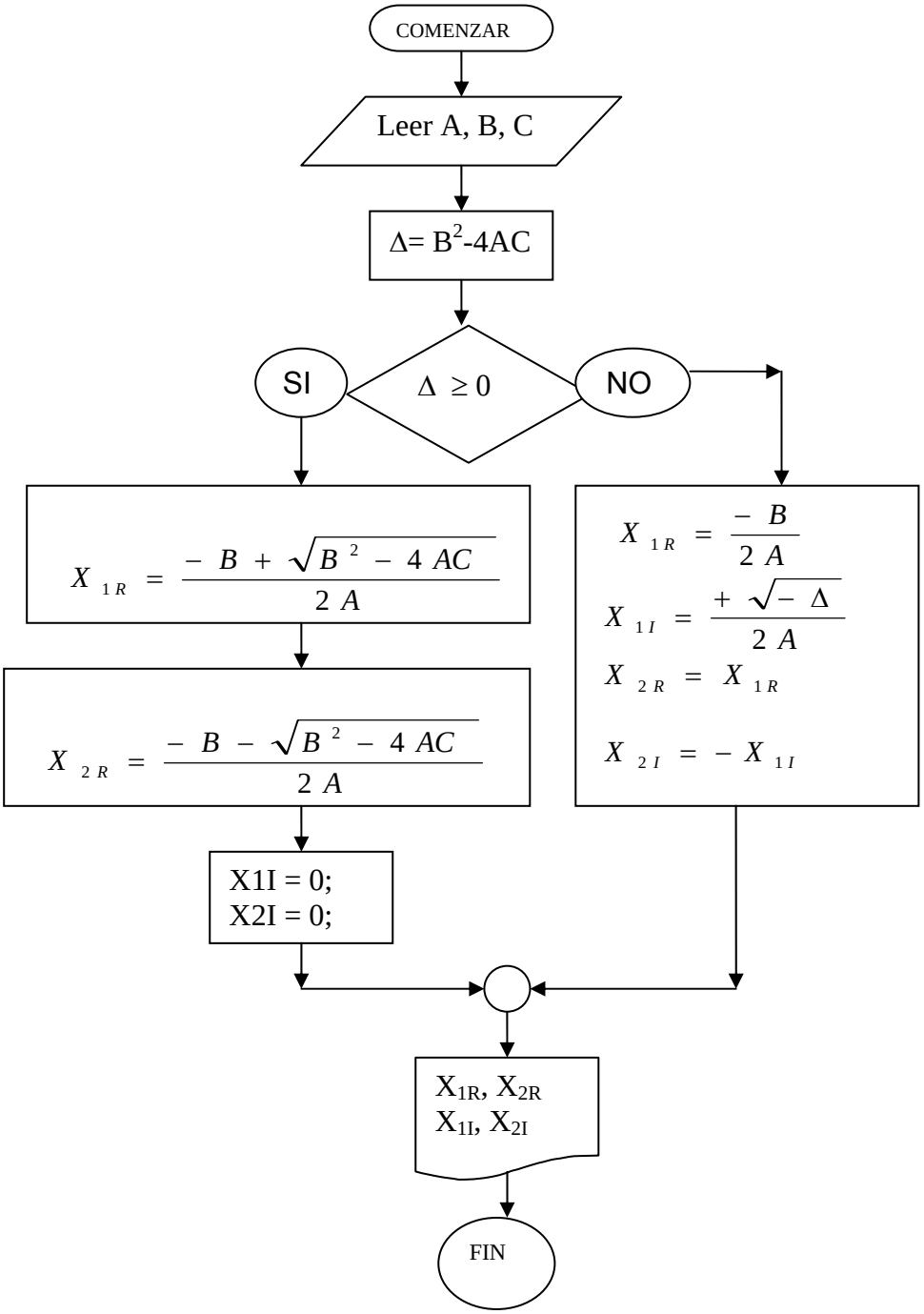
a) Resolución por estructura secuencial

Aquí sólo se considera, por defecto, el caso de raíces reales positivas, con discriminante (B^2-4AC) >0.

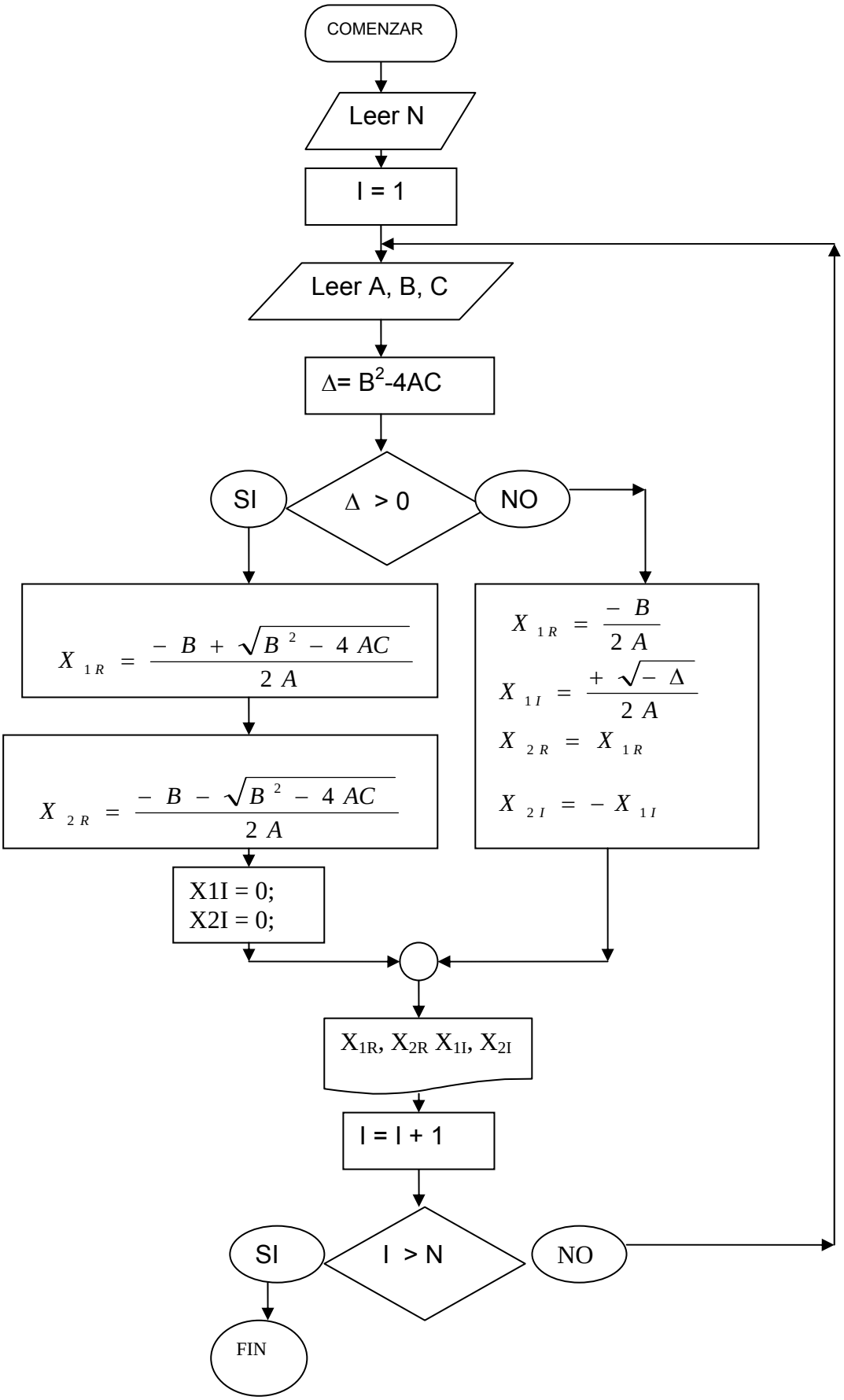


b) Resolución por estructura alternativa.

Para una terna dada de coeficientes A, B, C, este DDB considerará la posibilidad de que el discriminante pueda tener cualquier valor.

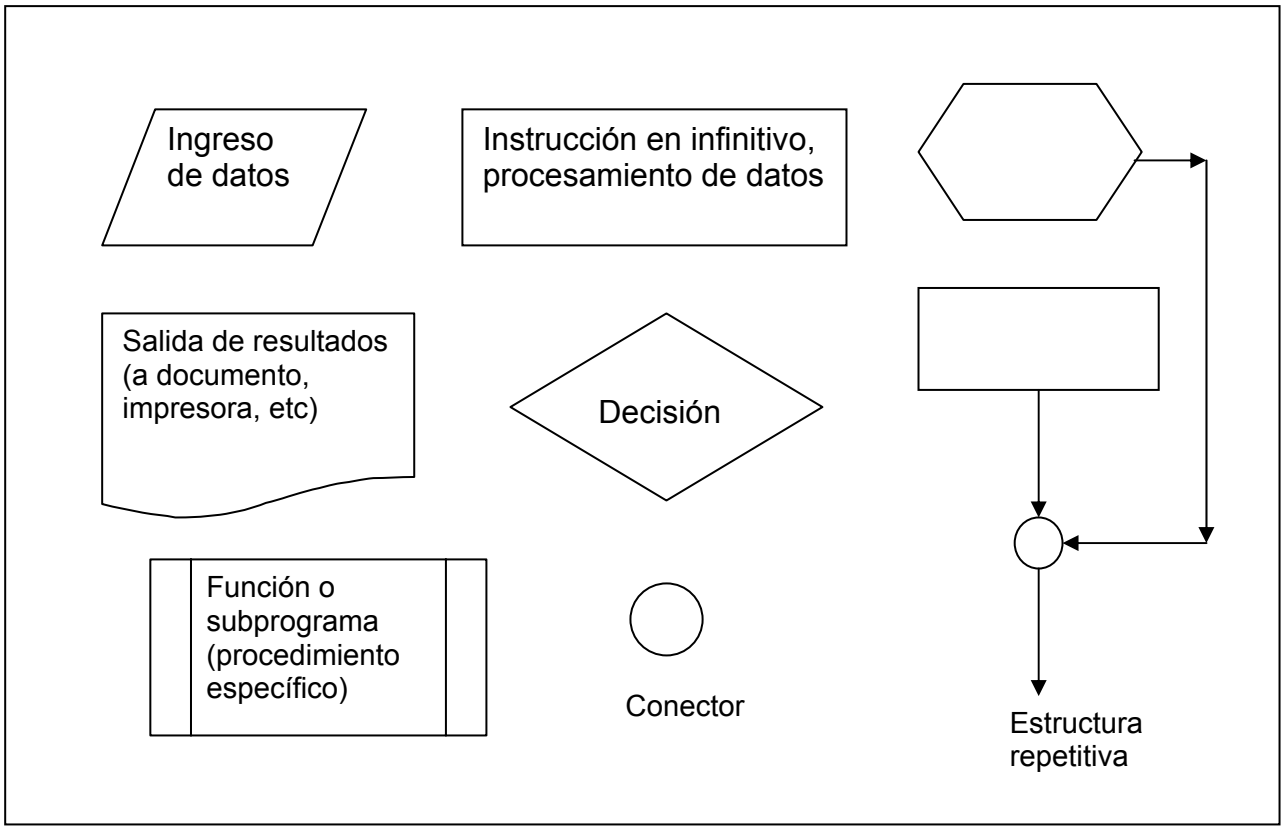


c) Resolución por estructura repetitiva: En este caso, se considerarán N ternas de coeficientes, A, B y C.



SIMBOLOS NORMALIZADOS

Aunque de los ejemplos dados se desprende la relación entre el tipo de símbolo y su función, listaremos aquí los símbolos normalizados más habituales. Los mismos se pueden obtener de la barra de herramientas "dibujo" del Word de Windows, en el botón "Autoformas"- "Diagramas de Flujo". Para inscribir texto dentro de ellos, apriete el botón derecho y elija "agregar texto".



LA SUMA EN DIAGRAMACION-PROGRAMACION

En el ejemplo de la estructura repetitiva, se observó la instrucción I = I + 1. Esta "instrucción", no tiene sentido como ecuación. No obstante, en programación significa

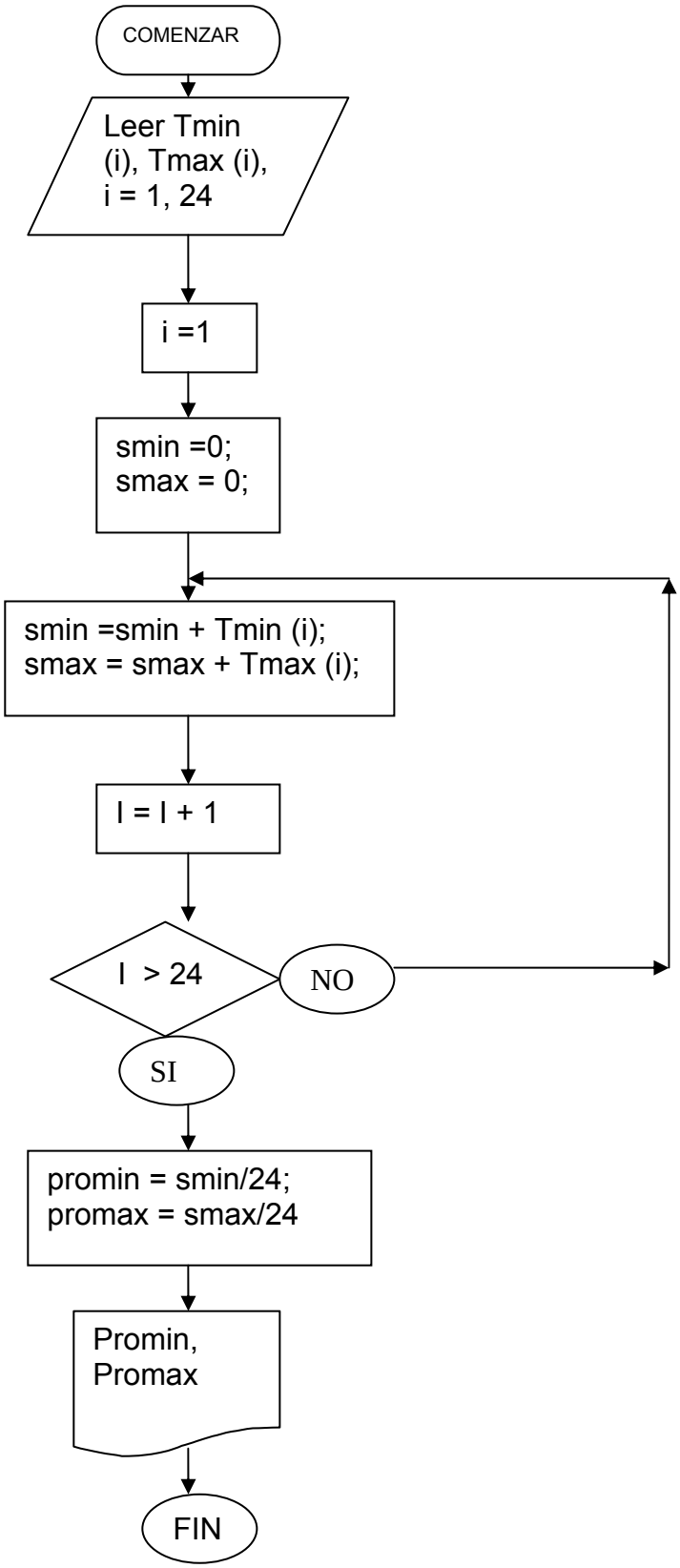
$$I\ posterior = I\ anterior + 1$$

Es decir, el "I anterior", es borrado del "casillero" (dirección) de memoria designado para la variable I, y es reemplazado por un valor igual al anterior + 1. En este sentido, realmente el signo igual en programación no coincide con su similar matemático, sino que significa realmente "asignación", es decir, "asignar a la variable escrita a la izquierda del signo igual, el resultado de la expresión evaluada a la derecha de ese signo". En consecuencia, aunque se usa el signo igual, la instrucción se entiende mejor así

$$I\ posterior \longleftarrow I\ anterior + 1$$

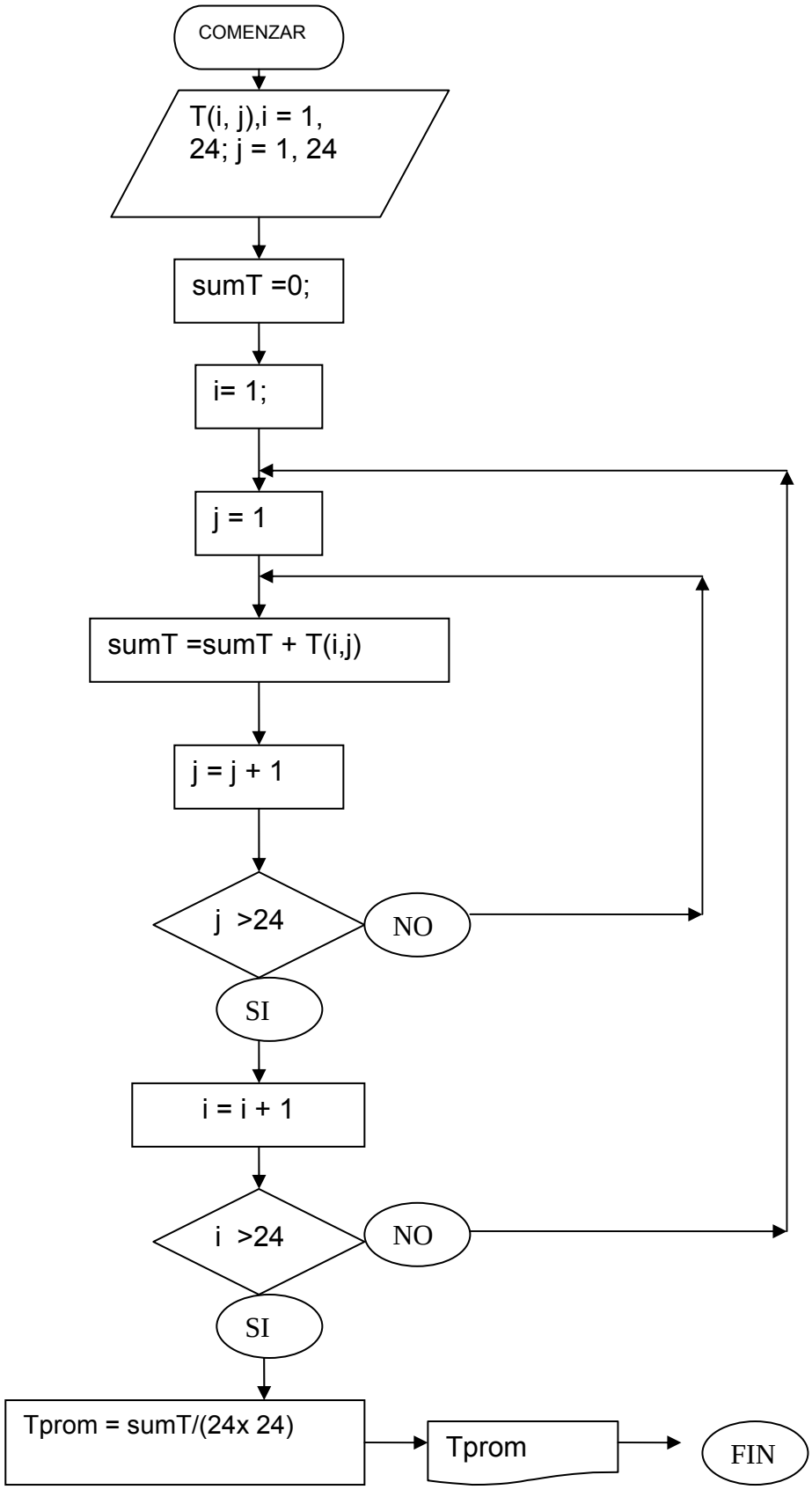
DIAGRAMAS DE BLOQUES UTILIZANDO VECTORES O MATRICES

Los "arreglos" (vectores, matrices) contienen elementos relacionados, designados con un nombre común. Por ejemplo, el SMN podría proveer diariamente las temperaturas mínimas y máximas de las Capitales Provinciales y de la Ciudad de Bs As, con dos vectores Tmin, y Tmax, c/u de 24 elementos. Si quisiéramos sacar el promedio de esos valores, el DDB sería:



DIAGRAMAS DE BLOQUES CON LAZOS REPETITIVOS ANIDADOS

Suponga Ud que las temperaturas de cada capital provincial de Argentina vienen dadas en forma horaria. Tendremos entonces la matriz, T(i, j), donde i = 1..24, y j = 1..24. Si se quiere sacar un promedio general, que incorpore todos los datos, podemos programar la siguiente estructura repetitiva.



Ejercicio de Autoevaluación: desarrollar un diagrama de Bloques que calcule las temperaturas promedio diarias de cada capital provincial, y, como en el Diagrama de Bloques anterior, la media general.

Finalmente, se darán aquí las etapas normalmente seguidas para resolver un problema de cierta magnitud y complejidad, mediante la computadora

ETAPAS DE LA RESOLUCION DE UN PROBLEMA MEDIANTE COMPUTADORA

1) Análisis del Problema
2) Obtención del método de resolución
3) Diagramación del método (DDB)
4) Codificación en lenguaje de alto nivel (Programación propiamente dicha)
5) Eliminación de errores (debugging)
6) Ejecución del programa y obtención de resultados. Representación gráfica
7) Análisis de los resultados. Si hay errores conceptuales, volver a 1) , 2 ó 4) . Si no, seguir en 8)
8) Refinamiento del programa (Mejoras en el diseño, legibilidad, documentación, actualización)

Estos conceptos se irán aclarando a medida que nos introduzcamos en el lenguaje de computadora específico, en nuestro caso, Matlab.

CAPITULO 2. ELEMENTOS PARA LA PLATAFORMA COMPUTACIONAL MATLAB

MATLAB es tanto un entorno de cálculo interactivo como un lenguaje de programación de alto nivel, cuya característica principal reside en trabajar en forma completamente matricial. Esta clase se divide en las partes siguientes:

- Introducción – Describe los componentes del sistema MATLAB.
- Entorno de Desarrollo – Introduce el entorno de desarrollo de MATLAB.

INTRODUCCIÓN

MATLAB es un lenguaje de alto rendimiento para computación en ciencias e ingeniería. Integra cálculo, visualización y programación en un entorno amigable donde los problemas y soluciones se expresan en notación matemática habitual. Los usos típicos de Matlab, son:

- Matemáticas y cálculo
- Algoritmos de desarrollo
- Modelado, simulación y desarrollo de prototipos
- Análisis de datos, exploración y visualización
- Graficación ingenieril y científica
- Desarrollo de aplicaciones **(programas propios del usuario)**, incluyendo la construcción de interfases gráficas para el usuario (GUI) para facilitar la entrada de datos y la visualización y análisis de los resultados.

MATLAB es un sistema interactivo, cuyo "dato básico" es un arreglo que no requiere dimensionamiento. Permite resolver problemas técnicos, especialmente aquellos con formulaciones de matriz y vector, en menos tiempo del que demandan otros lenguajes como C o Fortran.

El nombre de MATLAB proviene de "Matrix Laboratory". La Primera versión de Matlab la escribieron Cleve Moler y Jack Little en EEUU para dar acceso a la biblioteca de subprogramas científicos Fortran LINPACK y EISPACK (que programó Moler) a los alumnos de la Universidad de Nuevo México. Su idea era realizar un entorno integrado de programación y cálculo, poniendo especial énfasis en la productividad. Los dos fundaron la empresa de Matlab (Mathworks) en 1984.

MATLAB presenta una familia de programas que resuelven problemas específicos de un campo del conocimiento, llamados *Cajas de herramientas* (*toolboxes*). Las *toolboxes* están disponibles en áreas como estadística (con ajuste de curvas a datos), resolución de ecuaciones diferenciales, procesamiento de señal, sistemas de control, procesamiento de imágenes, etc. Nosotros esperamos que la capacidad que adquieran los alumnos en programación Matlab durante este curso les facilite luego, el uso de esos Toolboxes para eficientizar la programación de problemas de Cátedras de años superiores o durante su vida profesional.

El sistema MATLAB consta de cinco partes principales:

1) Entorno de desarrollo. Este representa el conjunto de herramientas y recursos que permiten el uso de las funciones y archivos MATLAB. Muchas de estas herramientas son interfases gráficas del usuario (GUI). Incluye el "escritorio" MATLAB y la ventana de comandos (Command Window), el historial de comandos, y "ventanas de visualización" (browsers) para ver la ayuda (help), la carpeta de trabajo, los archivos, y la ruta de acceso a información (search path).

2) La biblioteca de funciones matemáticas MATLAB (MATLAB Mathematical Function Library). Es una vasta colección de algoritmos matemáticos que va desde las funciones elementales como suma, seno, coseno, a funciones más sofisticadas como determinantes de matrices, autovalores, funciones de Bessel, y transformadas de Fourier.

3) El lenguaje MATLAB. Es un lenguaje de programación de alto nivel, de tipo matricial, con instrucciones de control de flujo, funciones preprogramadas y diversos tipos de datos. Asimismo, presenta varias alternativas para la entrada y salida de información y aspectos de programación orientada a objetos. Permite tanto la "programación en chico", para crear programas rápidos y descartables, y "programación en grande" para desarrollar aplicaciones sobre tareas de gran complejidad.

4) Realización de Gráficos. Es el sistema gráfico de MATLAB. Incluye comandos de alto nivel para visualización bi y tri-dimensional, procesamiento de imágenes, animación, y gráficos para presentaciones. También incluye comandos de bajo nivel que le permiten una total personalización de la apariencia de los gráficos, así como también para construir interfases gráficas de usuario (GUI) que dan funcionalidad y apariencia de "software" a los programas que Ud. realiza.

5) La interfase de Programas de Aplicaciones de MATLAB (MATLAB Application Program Interfase (API)). Esta biblioteca permite escribir programas C y Fortran que interactúen con MATLAB. Incluye recursos para llamar rutinas desde MATLAB (dynamic linking), llamando a MATLAB como "motor" computacional ("motor de cálculos") y para leer y escribir desde o en archivos MAT (generados por la ventana de comandos).

ENTORNO DE DESARROLLO

INTRODUCCIÓN

Este capítulo provee una introducción breve al ingreso y egreso de MATLAB, y a las herramientas y funciones que ayudan a trabajar con las variables y archivos de MATLAB.

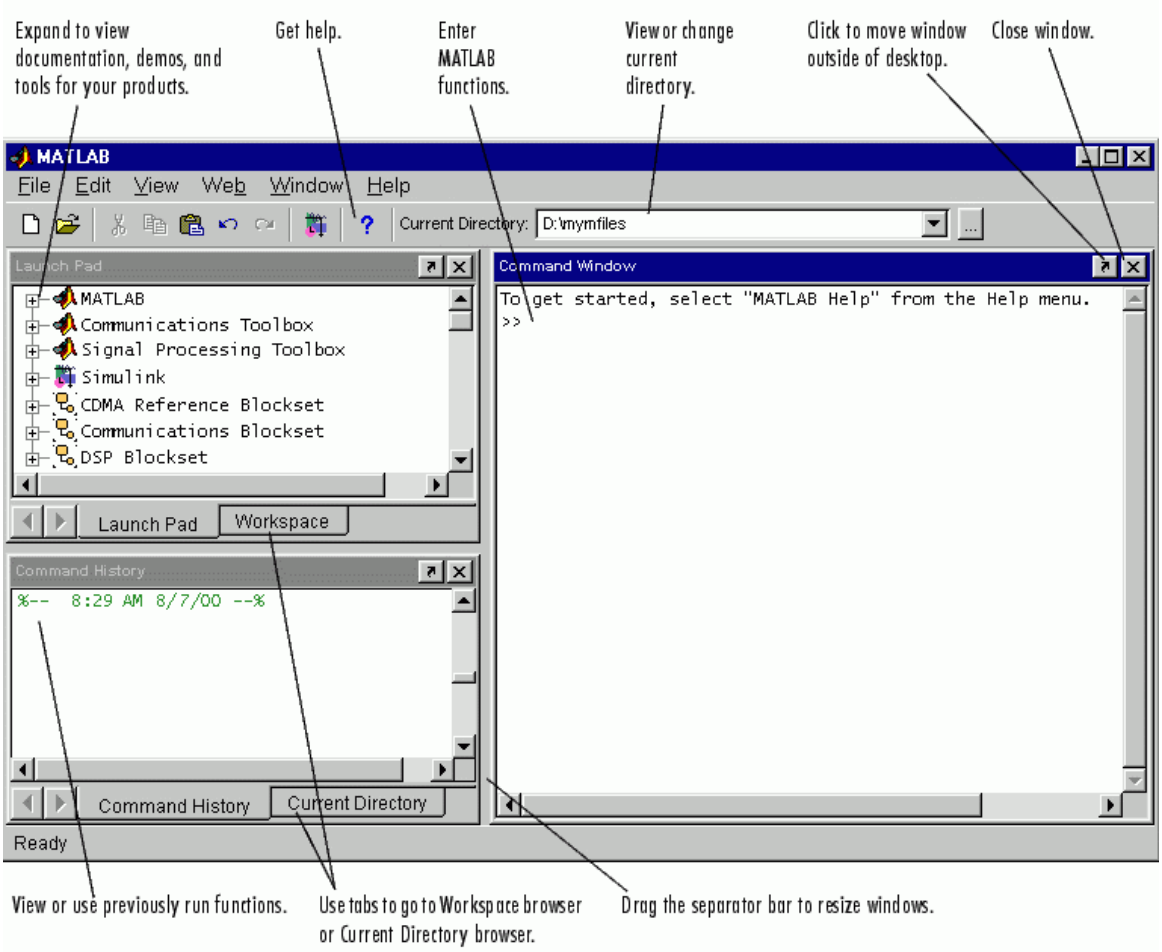

MATLAB 6.1.lnk

Para ingresar al programa se debe oprimir el ícono de Matlab en el escritorio o bien seleccionarlo en inicio-programas. Irse de Matlab es similar a hacerlo desde cualquier otra aplicación de Windows. Puede hacerlo con el botón "X" (cerrar) o bien desde el menú File, Exit-Matlab. Necesitará una computadora tipo Pentium II o superior con al menos 64 Mb de Memoria RAM.

ESCRITORIO MATLAB

Cuando comience MATLAB, el *Escritorio* MATLAB aparece, conteniendo herramientas (interfases gráficas de usuario) para el manejo de archivos, variables, y aplicaciones asociadas con MATLAB.

La primera vez que comienza MATLAB, el escritorio aparece como se muestra en la ilustración siguiente (Default View, apariencia por defecto), aunque su Launch Pad puede contener distintas entradas.



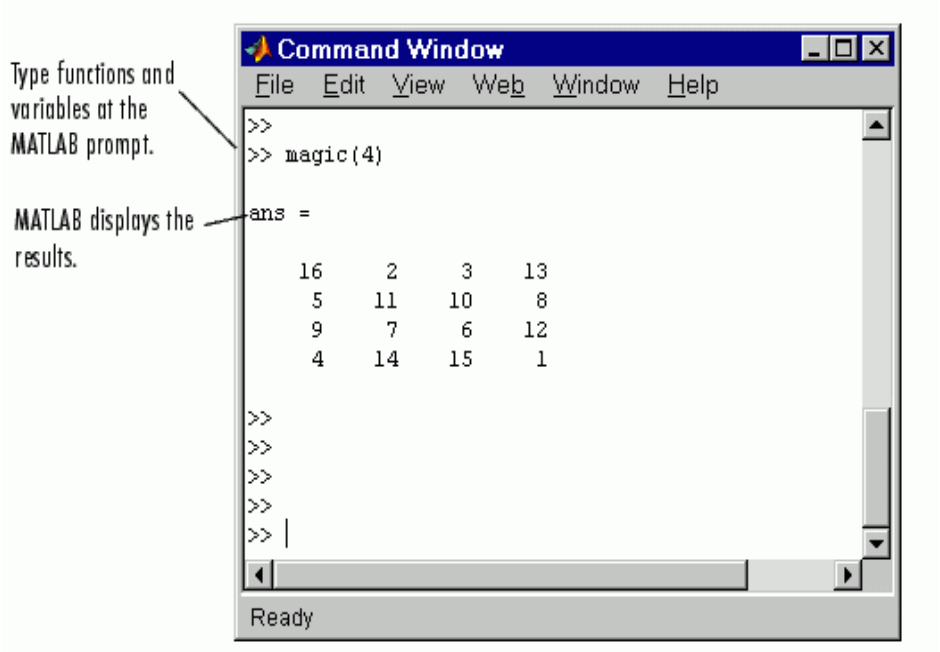
Se puede cambiar la apariencia del escritorio abriendo, cerrando y cambiando el tamaño de las herramientas en él. Las herramientas pueden moverse fuera del escritorio, o retornarlas otra vez dentro del escritorio (docking). Todas las herramientas del escritorio poseen aspectos comunes tales como menúes contextuales y atajos de teclado (keyboard shortcuts). Se pueden especificar ciertas características para las herramientas del escritorio seleccionando **Preferences** desde el menú **File**. Por ejemplo, se pueden especificar las características de la fuente (font) para el texto del Command Window.

HERRAMIENTAS DEL ESCRITORIO. Esta sección provee una introducción a las herramientas del escritorio de MATLAB. Tales herramientas son:

- La ventana de comandos (Command Window)
- El registro histórico de comandos (Command History Browser)
- La Plataforma de lanzamiento (Launch Pad)
- La ventana de ayuda (Help Browser)
- El directorio actual (Current Directory Browser)
- El espacio de trabajo (Workspace Browser)
- El editor de arreglos (vectores y matrices) (Array Editor)
- El editor de archivos y depurador de errores (Editor/Debugger)

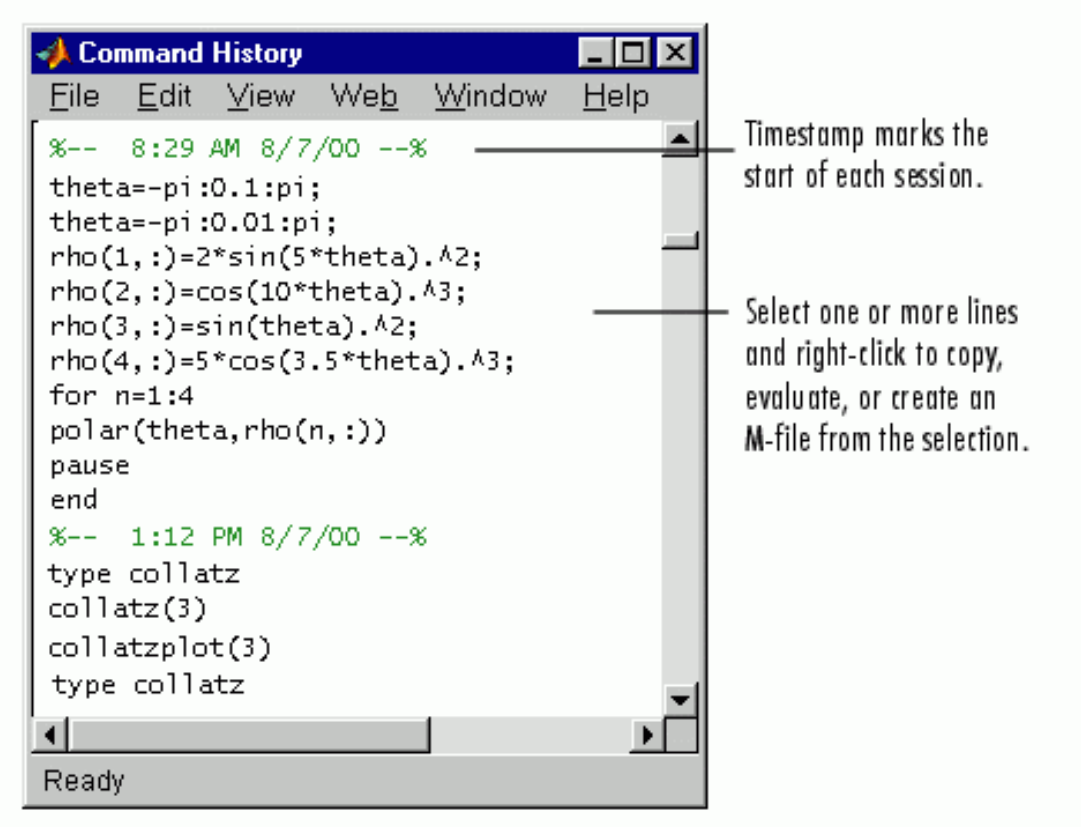
VENTANA DE COMANDOS (Command Window)

En la siguiente figura se observa el aspecto que posee la ventana de comandos. En esta ventana se realizan cálculos, se ejecutan funciones particulares de Matlab y se ejecutan programas escritos por los usuarios (archivos-M).



HISTORIAL DE COMANDOS (Command History)

Las líneas que se ingresan en la Command Window se graban en la ventana de la **Command History**. Allí, se pueden ver las funciones utilizadas previamente, y copiar y ejecutar líneas seleccionadas (Reutilizar código previamente escrito).



En esta ventana el registro de fecha y hora marca el comienzo de cada sesión.

Otra forma de registrar una sesión de MATLAB es utilizar la función *Diary*, que crea un archivo *.out. Se abre con “Diary” en el command window, y se cierra con “Diary off”.

EJECUCIÓN DE PROGRAMAS EXTERNOS

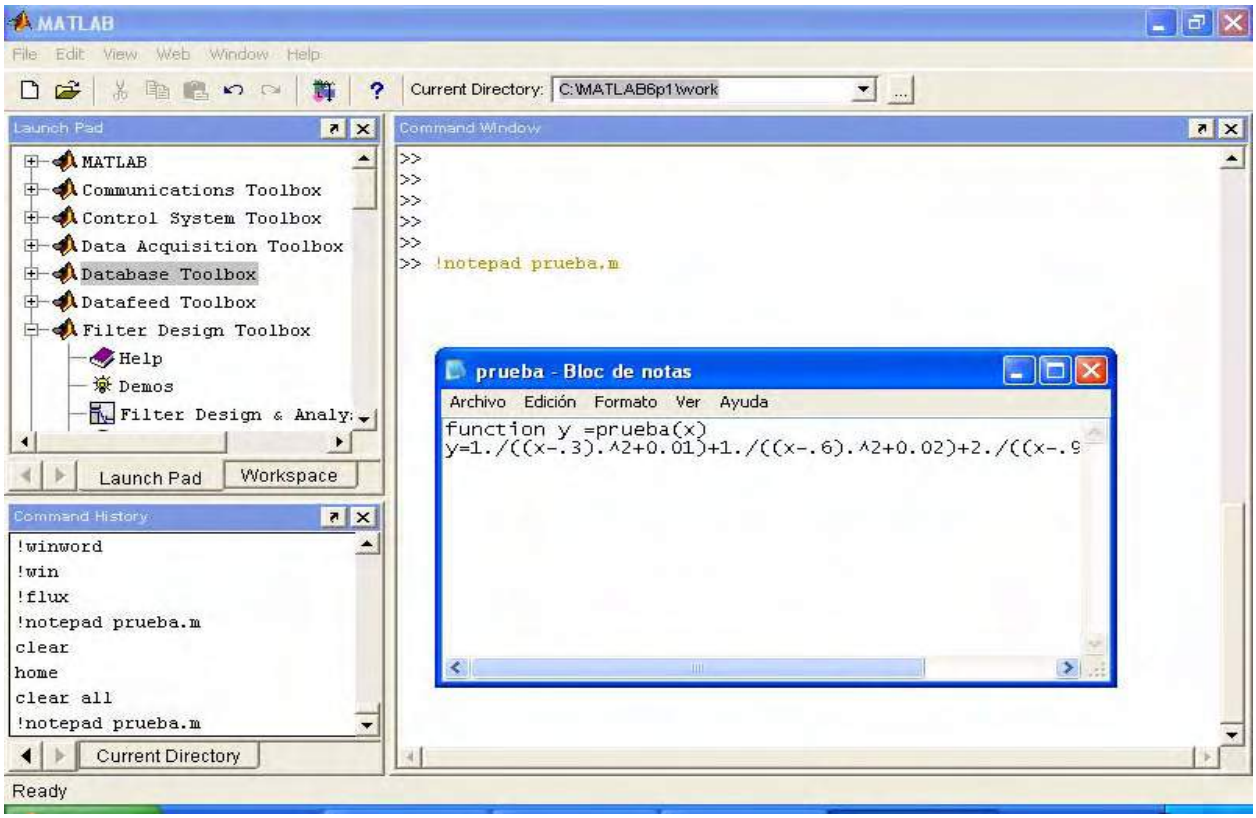
Una opción interesante consiste ende correr programas externos desde la ventana de comandos de MATLAB. El signo de exclamación ! es un *shell* e indica que el resto de la línea de entrada es un comando al sistema operativo. Esto resulta útil para invocar utilidades o correr otros programas sin salir de matlab. Por ejemplo, en Linux,

```
!emacs magik.m
```

invoca un editor llamado emacs para un archivo nombrado magik.m. Cuando se sale del programa externo, el control vuelve a MATLAB.

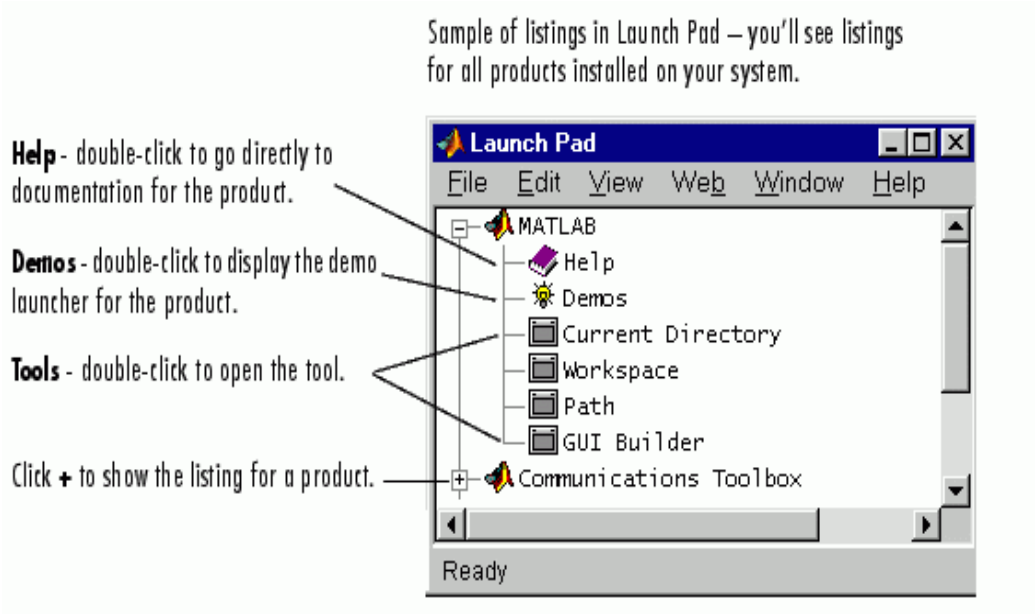
En el sistema operativo Windows, si se desea abrir un archivo prueba.m que contiene una función en lenguaje matlab, no con el editor propio sino con el bloc de notas (notepad), se puede hacer:

!notepad prueba.m



LAUNCH PAD

El Launch Pad es un "explorador de archivos" interno de MATLAB provee fácil acceso a herramientas, demos y documentación

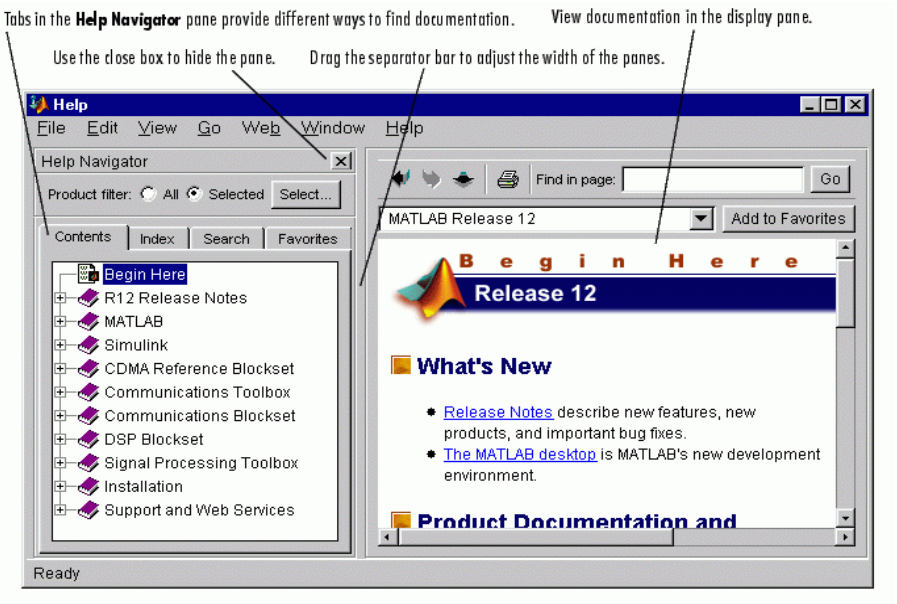


Las indicaciones en la figura muestran como acceder a documentación del producto, como acceder a ejecutar demos, como abrir herramientas, o como abrir subdirectorios que contienen información específica.

VENTANA DE VISUALIZACIÓN/ GESTIÓN DE AYUDA (Help Browser)

Usar el Help browser (to browse – "mirar sin modificar") se emplea para buscar y visualizar documentación para todos los productos Mathworks. El Help browser es un navegador de internet integrado en el escritorio de MATLAB que muestra documentos HTML .

Para abrir el Help browser, clickee el menú descolgable Help, o tipee helpbrowser en la Command Window.



El help browser consiste de dos paneles, uno de ellas el navegador (explorador interno) de ayuda (a la izquierda), que se puede utilizar para localizar la información, y el panel de visualización" (a la derecha) , donde se la observa. Se puede cerrar el de la izquierda, ampliar o reducir cada uno.

El navegador del help incluye:

- **Filtro de producto (Product filter)** – Configura la ayuda sólo para los productos que se especifiquen. Es muy conveniente restringir la ayuda a Matlab solamente, si no se usan otros programas asociados.
- **Solapa de Contenidos (Contents)** Visualiza los títulos y tablas de contenidos de la documentación de los productos. Es el explorador interno de la ayuda.
- **Solapa de Indice (Index)** – Encuentra entradas específicas (selected keywords) de la documentación de los productos.
- **Solapa de Búsqueda (Search)** – Busca una frase específica en la documentación. Para hallar la ayuda a una función específica, configure el tipo de búsqueda (Search type) a **Function Name**.
- **Solapa de Favoritos (Favorites)** – Muestra una lista de documentos de la ayuda, seleccionada previamente como favorita. Es muy útil irla confeccionando con el tiempo, para seleccionar rápidamente las ayudas más comúnmente solicitadas.

Una vez que encontró la localización de la ayuda en el Help Navigator, se puede leer en el cuadro de visualización. El mismo contiene distintos elementos que permiten:

- Mirar otras páginas - Use las flechas en la parte superior e inferior de las páginas, o los botones "forward" (hacia adelante) y "back" (hacia atrás) en la barra de herramientas.

- Señaladores (Bookmarks) – Cliquee el botón **Add to Favorites** de la barra de herramientas.

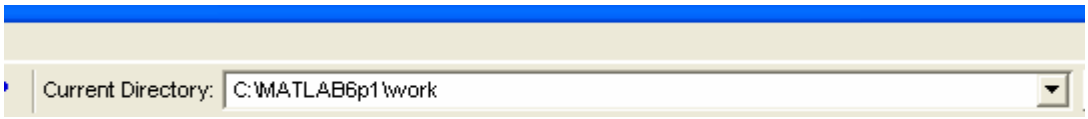
- Imprimir páginas – Cliquee el botón **print** en la barra de herramientas.

Si desea buscar alguna palabra o tema en la página cargada en el cuadro de visualización, se debe tipear en el campo permitido para **Find in page** de la barra de herramientas.

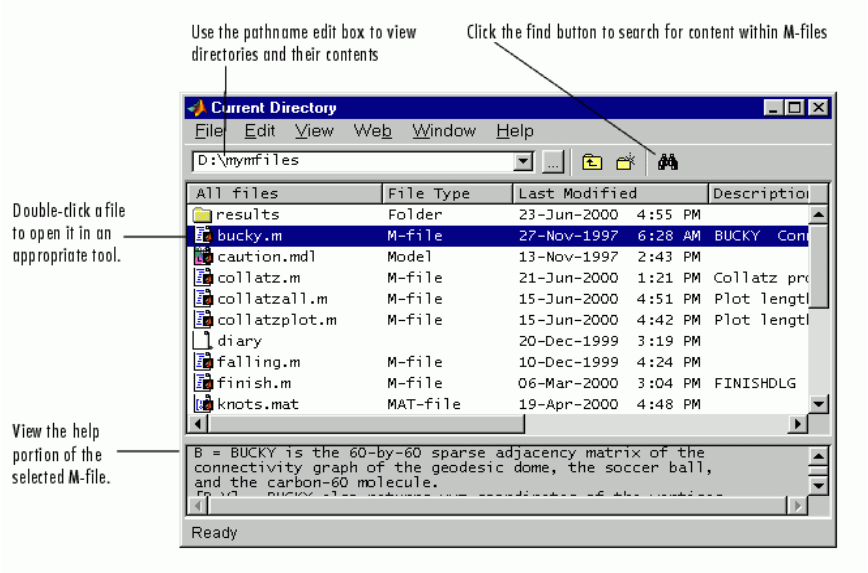
Para cerrar el nivel de ayuda elija el Menú “File”, y luego “Close Help".

VISUALIZACIÓN DEL DIRECTORIO ACTUAL (Current Directory Browser) Las operaciones con archivos de MATLAB usan el directorio actual (C:\MATLAB6p1\work) y la ruta de búsqueda (“search path”) como puntos de referencia. Cualquier archivo que se quiera ejecutar debe estar o en el directorio actual o en la search path. El path se visualiza en el Command Window luego de tipear “path”.

Para ver o cambiar rápidamente el directorio actual se debe utilizar la solapa del **Current Directory** en la barra de herramientas del escritorio, como se muestra a continuación:



Para buscar, ver, abrir o realizar cambios en los directorios y archivos relacionados con MATLAB, se debe usar el visualizador de directorio actual de MATLAB (Current Directory browser). Alternativamente, se pueden utilizar funciones dir, cd, y delete.



(Aquí el directorio de trabajo se llama D:\myfiles)

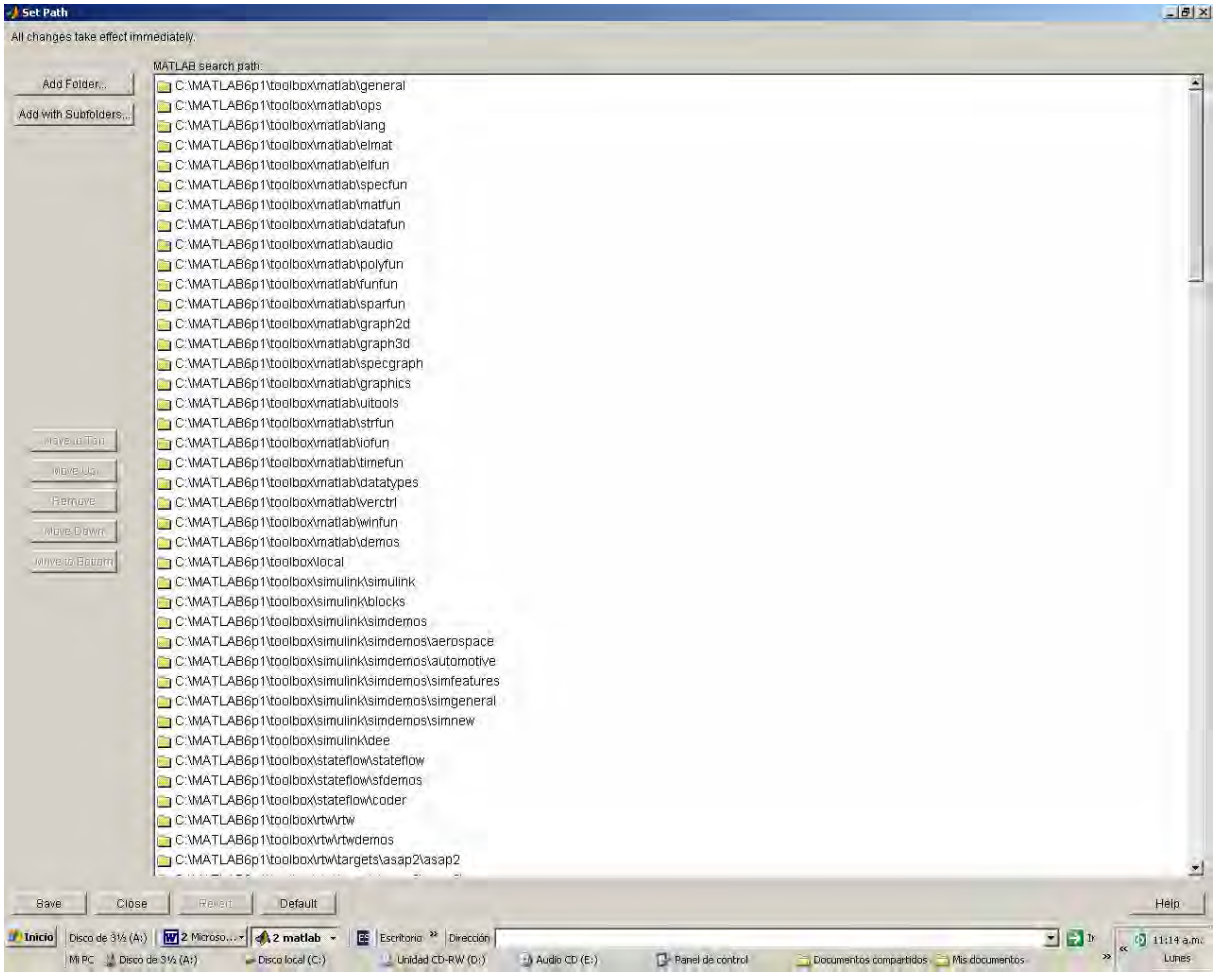
Aquí se observa la ventana del directorio actual (current directory) en forma aislada (undocked). Los archivos m, tanto programas contenidos originalmente en el software como aquellos realizados por el usuario, se abren cliqueando con el ratón.

Para integrarla de nuevo (dock) en la “default view” utilice el menú “view”, “dock”.

RUTA DE BÚSQUEDA (Search path)

Para ejecutar funciones o archivos m, llamados desde el Command Window en forma automática, MATLAB debe *saber* donde están, para esto utiliza el *search path*. En general, se utiliza el “search path” para incorporar archivos de las “cajas de herramientas” o toolboxes. El usuario puede también construir la suya

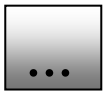
Para ver cuales directorios están en el search path o para cambiar la ruta de búsqueda, se debe seleccionar **Set Path** desde el menú **File** en el escritorio de MATLAB y utilizar el cuadro de diálogo **Set Path** .

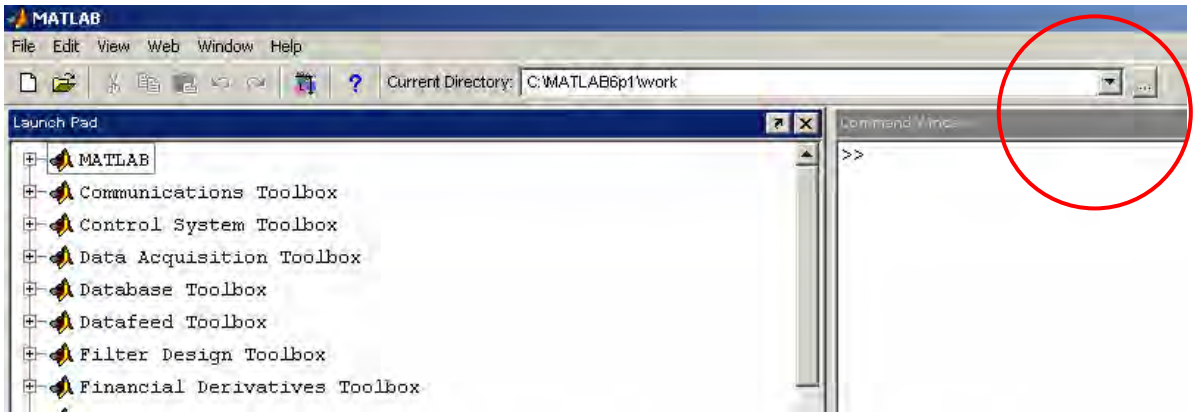


Oprimiendo el botón “add-folder” (agregar carpeta), el Matlab le ofrece un explorador de directorios para encontrar la ruta que desea incorporar. Luego de elegir, apriete “aceptar” en el cuadro insertado, y a posteriori “Save” en el cuadro de diálogo grande. Las acciones se pueden desacer con “revert” y todo puede volverse al estado inicial con “default”.

Selección del directorio de trabajo

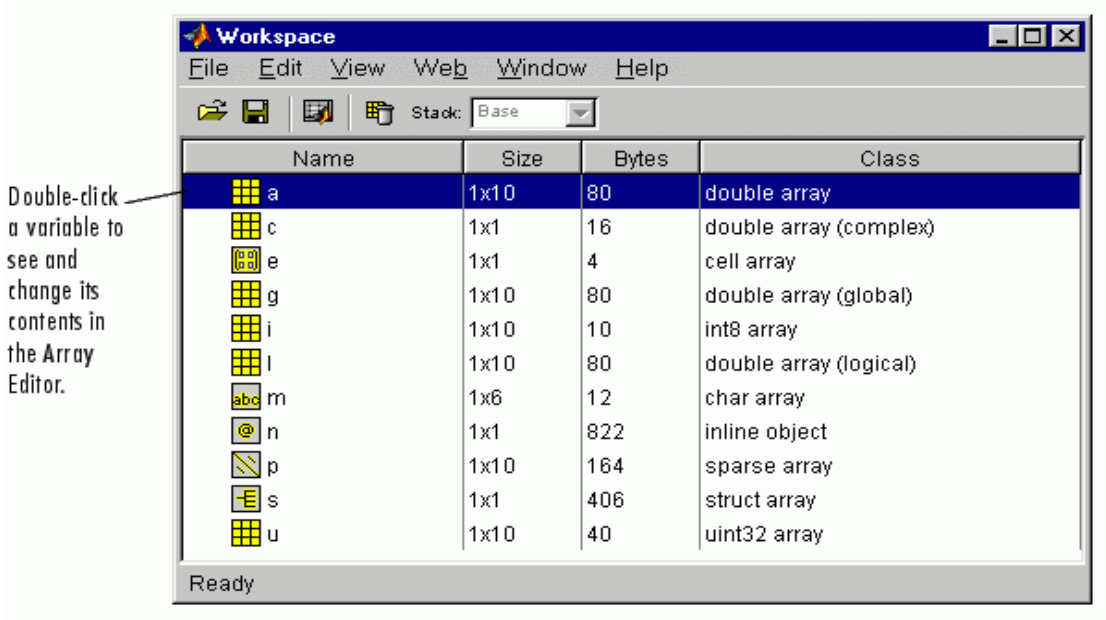
El Matlab trae por defecto el subdirectorio “work” para alojar los programas y sesiones de command window. Si el usuario desea usar temporariamente otro, sin dejar grabada esa ruta en matlab cuando lo apague (lo que es más cómodo que el “search path” para trabajo ocasional), se puede oprimir el botón que contiene tres puntos suspensivos, situado en la parte superior del escritorio de matlab, ligeramente corrido a la derecha.Oprimiendo ese botón, y buscando en el la estructura de directorios, se puede cambiar el directorio “work” a otro (que puede bien ser un subdirectorio de éste) para trabajar durante una sesión.





VISUALIZADOR DE LA CARPETA DE TRABAJO (Workspace Browser)

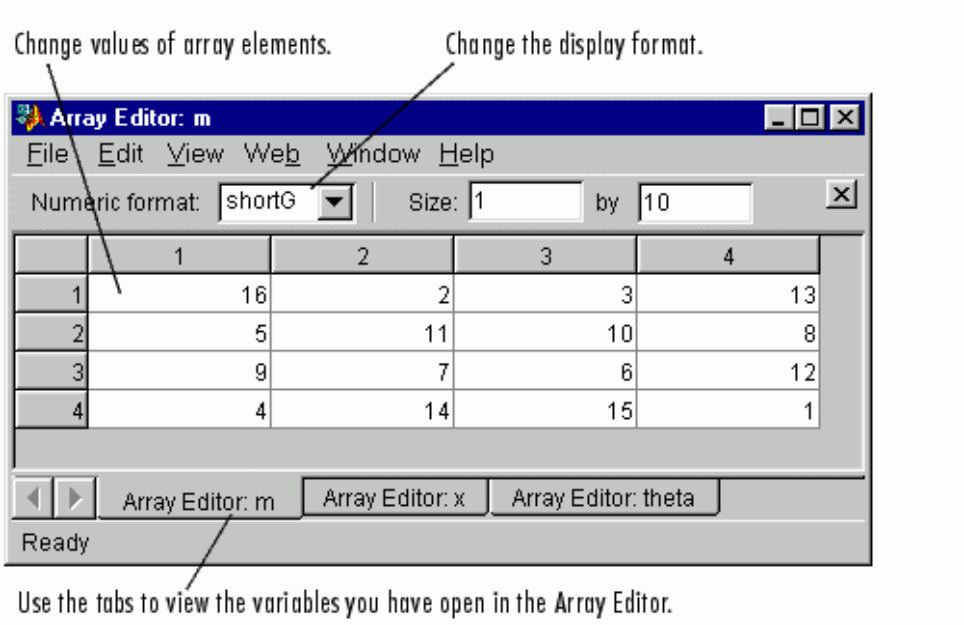
El espacio de trabajo MATLAB consiste de un conjunto de variables (arreglos con nombre) construidos durante una sesión de MATLAB y almacenados en memoria. Las variables se adicionan al espacio de trabajo a medida que se usan funciones, se corren archivos-M y se cargan espacios de trabajo ya grabados.
Para ver el espacio de trabajo y la información de cada variable, se emplea el Workspace browser, o bien las funciones who y whos.



Para borrar variables del espacio de trabajo, se selecciona la variable y luego se oprime **Delete** desde el menú **Edit.** Alternativamente, se puede utilizar la función clear .
El espacio de trabajo no se mantiene luego de terminarse la sesión MATLAB. Para guardar el workspace a un archivo que puede ser leído durante una sesión MATLAB posterior, seleccione **Save Workspace As** desde el menú **File**, o utilice la función save en el command window. Esto guarda el espacio de trabajo en un archivo binario, denominado archivo MAT, es decir, que tiene una extensión **.mat** . Hay opciones para guardar el workspace en distintos formatos. Para leer en un archivo-MAT, seleccione **Import Data** desde el menú **File** menu, o utilice la función load desde el command window.

EDITOR DE ARREGLOS (Array Editor)

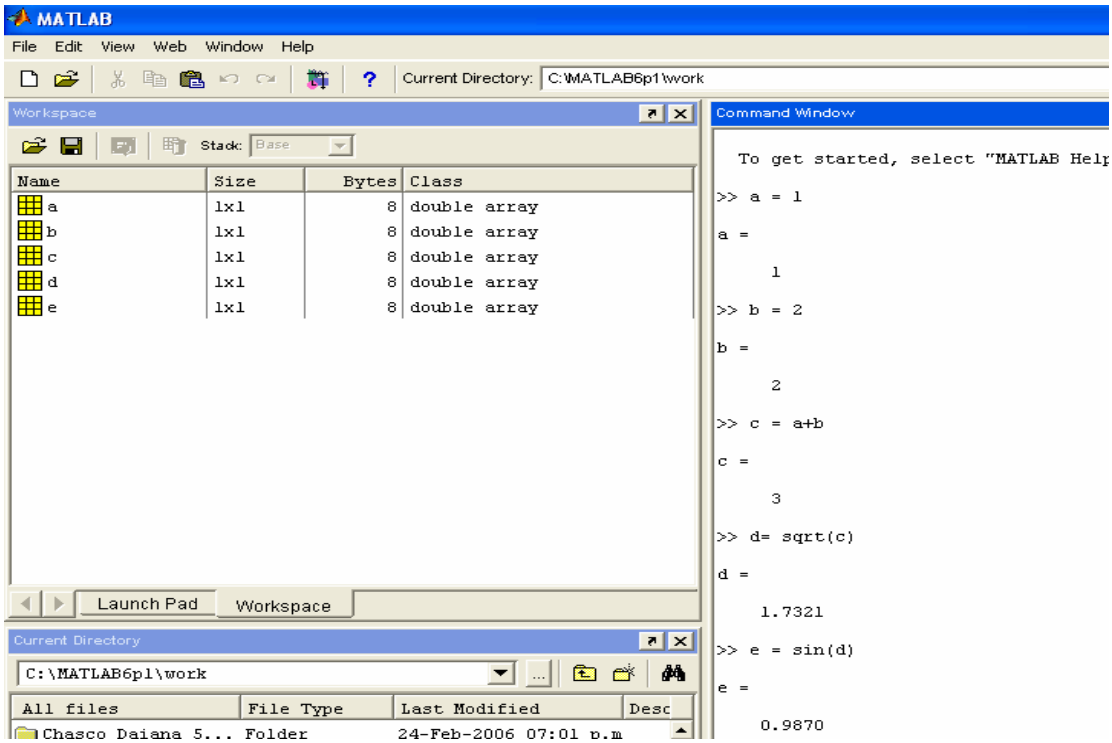
Haciendo doble-click en una variable del workspace browser aparce la ventana de edición de arreglos. Este editor permite ver y editar una representación visual de arreglos numéricos de una o dos dimensiones, cadenas alfanuméricas (strings), y arreglos de celdas de cadenas que se hallan en el espacio de trabajo (workspace).



Obsérvese que en esta figura se hallan abiertos tres arreglos, m, x y theta, pudiendo pasar de uno a otro arreglo seleccionando la solapa correspondiente. Esta ventana permite cambiar los valores de las variables, como así también modificar el formato numérico.

Ejemplo del uso de Matlab en modo interactivo

Aquí se da un ejemplo simple, donde se ingresa un valor a = 1, otro b = 2, se calcula "c" como la suma de ambos, se obtiene d como la raíz cuadrada de c, y e como el seno de d. Las variables que se van creando, que en este caso son arreglos de 1 x 1 (para matlab todas las variables son arreglos), se acumulan en el Espacio de trabajo (Workspace) y se visualizan a la izquierda.



El comando format

El comando format controla el formato numérico de los valores visualizados por MATLAB. **El comando afecta sólo la visualización de los números, no la forma en que MATLAB los calcula o los almacena**. Aquí se muestran los distintos formatos, junto a la salida resultante producida a partir de un vector x con componentes de distintas magnitudes.

Nota Para asegurar un adecuado espaciamiento, utilice una fuente de ancho constante (fixed-width font), tal como Fixedsys o Courier.

```
x = [4/3 1.2345e-6]
```

Entre otros, los formatos más típicos son

```
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
```

Si el elemento más grande de una matriz es mayor que 10^3 o menor que 10^{-3}, MATLAB aplica un factor de escala común para los formatos largos y cortos.
Además de los comandos de formato mostrados más arriba

```
format compact
```

suprime todas las líneas en blanco que aparecen en la salida. Esto conduce a ver más información sobre la pantalla o ventana.

Entrada de datos y Salida de resultados en Matlab interactivo

Suprimiendo la visualización de variables

Si se tipea una instrucción y se oprime **Enter**, MATLAB visualiza automáticamente los resultados en pantalla. Sin embargo, si se termina la línea con punto y coma, MATLAB realiza los cálculos pero no muestra la salida. Esto es muy útil para agilizar la ejecución de los programas, puesto que la visualización continua de resultados toma mucho tiempo de máquina.

Grabado de sesiones

Si al terminar de trabajar, o en cualquier momento de una sesión, se tipea el comando "save" en el command window, Matlab grabará la sesión desarrollada hasta ese momento, incluyendo las variables del espacio de trabajo en el archivo "Matlab.mat" Si quiere cambiar el nombre, puede hacer "save nombre", y grabará la sesión en el archivo nombre.mat. Luego, aunque haya apagado la computadora, podrá recuperarlos con load, o load nombre, respectivamente

Grabado selectivo de variables

Se pueden guardar no toda la sesión sino algunas variables en particular en forma selectiva y en archivos con nombre especificado por el usuario. Por ejemplo, el comando (sin comas entre los nombres de variables):

>> save nombre A x y

guarda las variables **A**, **x** e **y** en un archivo binario llamado ***nombre.mat .*** Para recuperarlas en otra sesión basta teclear:
>> load nombre

si se especifica, por ejemplo
>> **load nombre x y**
no se cargará la matriz A

El comando ***load*** admite las opciones ***-ascii*** y ***-mat***, para obligarle a leer en formato ASCII o binario, respectivamente.
El comando ***save*** permite guardar el estado de la sesión en formato ASCII utilizándolo de la siguiente forma (lo que va detrás del carácter % es un comentario que es ignorado por MATLAB):
>> save -ascii % almacena 8 cifras decimales
>> save -ascii -double % almacena 16 cifras decimales
>> save -ascii -double -tab % almacena 16 cifras separadas por tabs
aunque en formato ASCII sólo se guardan los valores y no otra información tal como los nombres de las matrices y/o vectores.
Cuando se recuperan estos archivos con *load….. -ascii* toda la información se guarda en una única matriz con el nombre del archivo. Esto produce un error cuando no todas las filas tienen el mismo número de elementos.
Con la opción ***-append*** en el comando ***save*** la información se guarda a continuación de lo que hubiera en el archivo.

CAPITULO 3. NOTAS SOBRE LA MANIPULACION DE MATRICES EN MATLAB

En MATLAB, una matriz es un arreglo rectangular de números. Se le asigna un significado especial a las matrices de 1 x 1, los escalares, y a las matrices con una sola fila o columna, los vectores. MATLAB tiene otras formas de almacenar datos numéricos y no numéricos, pero siempre es preferible comenzar asumiendo que todo es una matriz. Las operaciones en MATLAB se diseñan para ser tan naturales como sea posible. Mientras otros lenguajes de programación trabajan con un número a la vez, MATLAB permite trabajar con matrices enteras con facilidad..

INGRESO DE MATRICES

Una buena forma de empezar con MATLAB ***(MATrix LABoratory)*** es aprendiendo como operar con las matrices. En primera instancia se deben ingresar los valores que contiene las filas y las columnas. Una forma de hacerlos es manual, por ejemplo:

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

Luego, MATLAB muestra la matriz que se ha ingresado.

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

Una vez que se ingresó la matriz, ésta automáticamente queda grabada, en el espacio de trabajo (workspace). Para recuperar sus valores, es decir para usarlos, sólo tiene que tipear ***A***.

La matriz que se ha ingresado previamente se denomina mágica por sus curiosas características: la suma de los elementos de filas y columnas, y de los de las dos diagonales principales, dará siempre el mismo resultado. Estas propiedades se verifican en la próxima sección.

SUMA, TRANSPUESTA Y DIAGONAL

Matlab posee la función sum para realizar sumas. En el caso de las matrices, con la aplicación de esta función se suman los valores contenidos en las columnas. Si aplicamos esta función a la matriz A:

```
>>sum (A)
```

MATLAB responde con

```
ans =
   34   34   34   34
```

Cuando no se especifica una variable para asignar el resultado, MATLAB usa la variable ans, abreviatura de *answer*, para almacenar los resultados de un cálculo. Se calculó un vector fila conteniendo las sumas de las columnas de A. Cada una de ellas da la misma suma, 34.

¿Que sucede con las sumas de las filas? MATLAB tiene una preferencia para trabajar con las columnas de una matriz, así que la forma más fácil de realizar las sumas por fila es obtener la transpuesta de la matriz, calcular la suma de las columnas de la transpuesta, y luego transponer el resultado. La operación de la transpuesta se representa mediante un apóstrofe. En tal operación, la matriz es reposicionada alrededor de su diagonal principal, y transforma un vector fila en un

vector columna. Así

```
>>A'

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

mientras que

>>sum(A')'

produce un vector columna conteniendo la suma de las filas originales

```
ans =
   34
   34
   34
   34
```

La función diag genera un vector con los elementos de la diagonal principal.

```
>>diag(A)
```

dando lugar a

```
ans =
   16
   10
    7
    1
```

En consecuencia, sumando los mismos

>>sum(diag(A))

se obtiene

```
ans =
   34
```

SUBÍNDICES

Los elementos de la fila i y columna j de A se denotan por A(i,j). Por ejemplo, A(4,2) es el número de la cuarta fila y segunda columna. En nuestro cuadrado *mágico*, A(4,2) es 15. Es posible también referir los elementos de una matriz con sólo subíndice, A(k). Esta es la forma usual de referenciar los vectores fila y columna. Pero puede usarse también en una matriz bidimensional, en cuyo caso el arreglo se visualiza como un largo vector columna, formado por las columnas de la matriz original. Así, en nuestro cuadrado mágico, A(8) es otra forma de nombrar el valor 15 almacenado en el elemento(4,2).

Si se usa un valor de elemento fuera de la matriz, es un error

```
>>t = A(4,5)

Index exceeds matrix dimensions.
```

Por otro lado, si se almacena un valor en un elemento que está fuera de la matriz, el tamaño de la misma se incrementa automáticamente para incorporar al recién llegado.

```
X = A;
```

```
X(4,5) = 17

X =
   16     3     2    13     0
    5    10    11     8     0
    9     6     7    12     0
    4    15    14     1    17
```

EL OPERADOR DOS PUNTOS ":"

Los *dos puntos* **:** están entre los operadores más importantes de MATLAB. Se emplea para generar variables de intervalo. Aparecen aplicado en situaciones diversas.

La expresión

```
>>x=1:10
```

es un vector fila conteniendo los enteros de 1 a 10

```
1    2    3    4    5    6    7    8    9    10
```

para obtener un espaciado no unitario, se puede especificar un incremento. Por ejemplo,

```
>>x=100 : -7 :50
```

es

```
100    93    86    79    72    65    58    51
```

Se puede utilizar un espaciado no entero también

```
>>x=0 : pi/4 : pi
```

da lugar a

```
0    0.7854    1.5708    2.3562    3.1416
```

Pueden aparecer en expresiones de subíndices que indiquen partes de una matriz. Por ejemplo:

```
>>A(1:10,j)
```

Denota los primeros 10 elementos de la columna j-ésima de A. Así

```
>>sum(A(1:4,4))
```

calcula la suma de la cuarta columna. El operador dos puntos por sí mismo denota *todos* los elementos en una fila o columna de una matriz , y la palabra clave (keyword) end indica la ultima fila o columna. Así:

```
>>sum(A(:,end))
```

calcula la suma de los elementos de la última columna de A, empleando todas las filas.

```
ans =
    34
```

GENERANDO MATRICES

Esta sección incluye otras formas de crear matrices

MATLAB provee cuatro funciones para generar matrices básicas.

zeros	Todos ceros
ones	Todos unos
rand	Elementos aleatorios uniformemente distribuidos
randn	Elementos aleatorios normalmente distribuidos
magic	Genera matrices mágicas

Aquí se dan algunos ejemplos.

```
>>Z = zeros(2,4)

Z =
    0    0    0    0
    0    0    0    0

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

F =
    5    5    5
    5    5    5
    5    5    5

>>N = fix(10*rand(1,10))
N =
    4    9    4    4    8    5    2    6    8    0
```

(genera una matriz de una fila y diez columnas, con números aleatorios entre 0 y 1, multiplica por diez elemento a elemento, y luego trunca la parte entera de cada uno.

```
>>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
```

La matriz RANDN(N) es una matriz de N x N con entradas al azar, elegidas de una distribución normal con promedio cero, y desviación típica igual a la unidad.

MATRIZ VACIA

Se pueden borrar filas y columnas a partir de una matriz usando sólo un par de corchetes. Comenzar con

```
>>X = A; % A es la matriz "mágica" de 4 x 4
```

Luego, para borrar la segunda columna de X, usar

```
>>X(:,2) = [ ]
```

Esto cambia X a

```
X =
   16    2   13
    5   11    8
    9    7   12
    4   14    1
```

Si se borra un solo elemento de la matriz, el resultado será más una matriz, de manera que las expresiones tales como

```
>>X(1,2) = [ ]
```

Resultan en error.

Sin embargo, el uso de un solo subíndice borra un elemento individual, o una secuencia de elementos, y da nueva forma al arreglo ("reshape") transformándolo en un vector fila. Así

```
X(2:2:10) = [ ]
```

resulta en

```
X =
   16    9    2    7   13   12    1
```

ÁLGEBRA LINEAL

Informalmente, los términos *matriz* y *arreglo* se usan a menudo en forma indistinta. Más precisamente, una *matriz* es un arreglo numérico bidimensional que representa una transformación lineal. Las operaciones matemáticas que definen matrices son el objeto del *álgebra lineal*.

Sea el cuadrado mágico de Dürer

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

provee varios ejemplos que dan una buena muestra de las operaciones matriciales que realiza MATLAB.

El símbolo de multiplicación, *, denota la *multiplicación de matrices* que se lleva a cabo entre filas y columnas. Por ejemplo:

```
>>A'*A

ans =
   378   212   206   360
   212   370   368   206
   206   368   370   212
   360   206   212   378
```

El determinante de esta matriz particular es cero en este caso, indicando que la matriz es singular.

```
>>d = det(ans)

d =
     0
```

Volviendo al cuadrado mágico A, sus autovalores son interesantes.

```
>>e = eig(A)

e =
   34.0000
    8.0000
    0.0000
   -8.0000
```

Uno de los autovalores es cero, lo cual es otra consecuencia de la singularidad. El máximo autovalor es 34, la suma mágica. Esto es así porque el vector de todos los unos es un autovector.

El polinomio característico del cuadrado mágico es

```
>> poly (A)

ans =

  1.0e+003 *

    0.0010   -0.0340   -0.0800    2.7200   -0.0000
```

Para realizar el producto de matrices elemento a elemento se debe colocar un punto a una de las matrices, por ejemplo:

```
A .* A
```

>>el resultado es un arreglo conteniendo los cuadrados de los enteros de 1 a 16 en un orden inusual.

```
ans =
   256     9     4   169
    25   100   121    64
    81    36    49   144
    16   225   196     1
```

Dentro de las operaciones que se realizan entre matrices se debe incluir la divisón izquierda \. Esta operación merece una explicación especial. Supongamos el siguiente sistema de ecuaciones lineales:

```
>>A X=B
```

Donde x y b son vectores columna y A es una matriz cuadrada invertible. La solución de este sistema de ecuaciones lineales se puede escribir así:

Esto significa que el operador barra invertida premultiplica por la inversa de la matriz A. **Este operador constituye una excelente herramienta para resolver rápidamente sistemas de ecuaciones lineales.**

```
>>X=inv(A)*B   ó
   X=A\B
```

CONSTRUYENDO TABLAS

Las operaciones con arreglos son útiles para construir tablas. Supone que n es el vector columna

```
>>n = (0:9)';
```

Luego

```
>>pows = [n  n.^2  2.^n]
```

construye una tabla de cuadrados con potencias de dos.

```
pows =
    0     0     1
    1     1     2
    2     4     4
    3     9     8
    4    16    16
    5    25    32
    6    36    64
    7    49   128
    8    64   256
    9    81   512
```

Las funciones matemáticas elementales operan con arreglos elemento a elemento, de forma que:

```
format short g
>>x = (1: 0.1: 1.6)';
l>>ogs = [x log10(x)]
```

construye una tabla de logaritmos.

```
 logs =
    1.0         0
    1.1   0.04139
    1.2   0.07918
    1.3   0.11394
    1.4   0.14613
    1.5   0.17609
    1.6   0.20412
```

OTRAS ESTRUCTURAS DE DATOS

Esta sección lo introduce en algunas estructuras de datos en MATLAB, incluyendo:
Arreglos multidimensionales
Arreglos de celdas
Caracteres y texto
Estructuras

ARREGLOS MULTIDIMENSIONALES

Los arreglos multidimensionales en MATLAB son arreglos con más de dos subíndices. Se pueden crear llamando a zeros, ones, rand, o randn con más de dos argumentos, por ejemplo,

>>R = randn(3,4,5);

Crea un arreglo de 3x 4x 5 es decir, con 60 elementos aleatorios normalmente distribuidos.
Un arreglo tridimensional podría representar datos físicos tridimensionales, por ejemplo la temperatura en un salón, muestreada según una grilla. O bien podría representar una secuencia de matrices, $A^{(k)}$, o muestras de una matriz dependiente del tiempo, $A(t)$. En estos últimos casos, el elemento (i, j) de la matriz kth , or la matriz t_k, se denota con A(i,j,k).

ARREGLO DE CELDAS

Los arreglos de celdas en MATLAB ("cell arrays") son arreglos multidimensionales cuyos elementos son copias de otros arreglos. Se puede crear un arreglo de celdas de matrices vacías con la función cell . Pero, con mayor frecuencia, se crean arreglos de celdas delimitando una colección miscelánea de "cosas" entre llaves, {}. Las llaves se usan, asimismo, con subíndices para acceder los contenidos de varios arreglos.

Por ejemplo,

```
>>C = {A  sum(A)  prod(prod(A))}
```

Produce un arreglo de celdas de 1 x 3. Estas tres celdas contienen el cuadrado mágico, el vector fila de la suma de los elementos de las columnas, y el producto de los productos de los elementos de las columnas, que en definitiva es un número. Cuando se muestra C, se observa

```
C =
   [4x4 double]    [1x4 double]    [20922789888000]
```

Como las primeras dos celdas son muy grandes para imprimirlas en este espacio limitado, se las representa con nomenclatura. La tercer celda se puede incluir porque incluye un sólo número (16!) que se puede mostrar.

Aquí se observan dos puntos importantes para recordar.

Primeramente, para "recuperar" los contenidos de alguna de las celdas, se pueden utilizar subíndices entre llaves. Por ejemplo, C{1} trae el valor del cuadrado mágico y C{3}vale 16!. Segundo, los arreglos de celdas contienen *copias* de otros arreglos, no *punteros* de otros arreglos. Si se cambia A, por ejemplo, nada sucede con C (mientras no se calcule ex profeso otra vez, no hay actualización automática, en este caso el mecanismo no es como el de una planilla de cálculo) .

Se pueden emplear arreglos tridimensionales para almacenar una secuencia de matrices del *mismo* tamaño, y arreglos de celdas para guardar una secuencia de matrices de *distinto* tamaño.

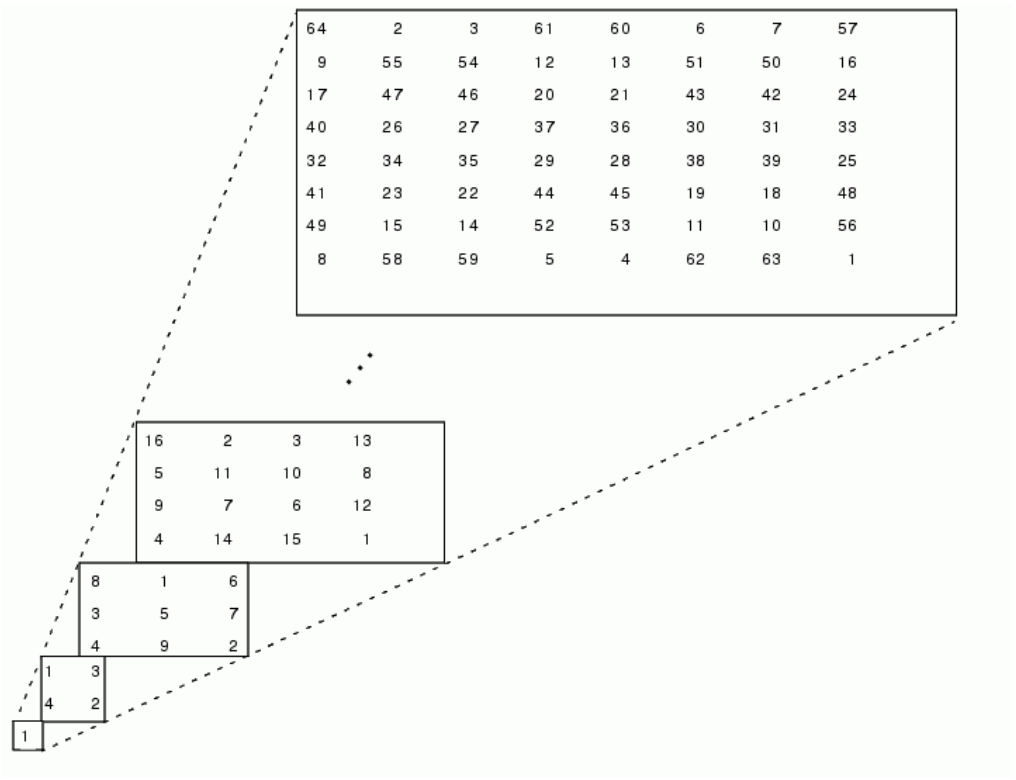
Por ejemplo,

```
>>M = cell(8,1);
for n = 1:8
  M{n} = magic(n);
end
M
```

Produce una secuencia de cuadrados mágicos de distinto orden.

M =

```
[          1]
[ 2x2  double]
[ 3x3  double]
[ 4x4  double]
[ 5x5  double]
[ 6x6  double]
[ 7x7  double]
[ 8x8  double]
```



Se puede recuperar, el cuadrado mágico de 4 x 4, haciendo
>>M{4}
Para obtener
ans =

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

CARACTERES Y TEXTO

Se puede ingresar texto en MATLAB utilizando apóstrofes. Por ejemplo,

s = 'Hola'

El resultado no es del mismo tipo de arreglo numérico que hemos tratado hasta ahora. Es un arreglo de caracteres de 1x 5.

Internamente, los caracteres se almacenan como números.

La instrucción

a = double(s)

convierte el arreglo de caracteres en una matriz numérica que contiene representaciones de los códigos ASCII de cada carácter. El resultado es

a =
104 111 108 97

s = char(a)
revierte la conversión.

El convertir números en caracteres permite investigar las distintas fuentes disponibles en su computadora. Los caracteres imprimibles se representan en el conjunto básico de caracteres ASCII por los enteros 32:127. Los enteros menores de 32 representan caracteres no imprimibles. Estos caracteres se pueden ordenar apropiadamente en un arreglo de 6 x 16 mediante la función

F = reshape(32:127,16,6)';
Que da nueva forma ("reshape") a una lista de números de 32 a 127 como una matriz de 16 x 6.
Los caracteres imprimibles en el conjunto extendido de caracteres ASCII (American Standard Code for Information Interchange) se representan por F+128. Cuando estos enteros se interpretan como caracteres, el resultado depende de la fuente (font) particular que está siendo utilizada. Si se tipean las sentencias

```
>>char (F)
ans =
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~□

>> char (F+128)
ans =
 ¡¢£¤¥¦§¨©ª«¬-®¯
°±²³´µ¶·¸¹º»¼½¾¿
ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏ
ÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
àáâãäåæçèéêëìíîï
ðñòóôõö÷øùúûüýþÿ
```

La concatenación con corchetes combina variables de texto para formar cadenas más grandes. La sentencia

>>h = [s, ' mundo']

combina las cadenas horizontalmente y produce

h =
Hola mundo

Obsérvese si ahora se los quiere poner en columna

```
>> v = [s; 'mundo']

??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.

>>  s= 'hola '

s =

hola

>> v = [s; 'mundo']

v =

hola
mundo
```

CAPITULO 4. GRÁFICOS EN MATLAB.

REPRESENTACION BÁSICA

MATLAB posee excelentes recursos para visualizar vectores y matrices como gráficos, así como para editar, documentar, e imprimir estos gráficos. Esta sección describe algunas funciones importantes y provee ejemplos de algunas aplicaciones típicas.

GRAFICOS BIDIMENSIONALES

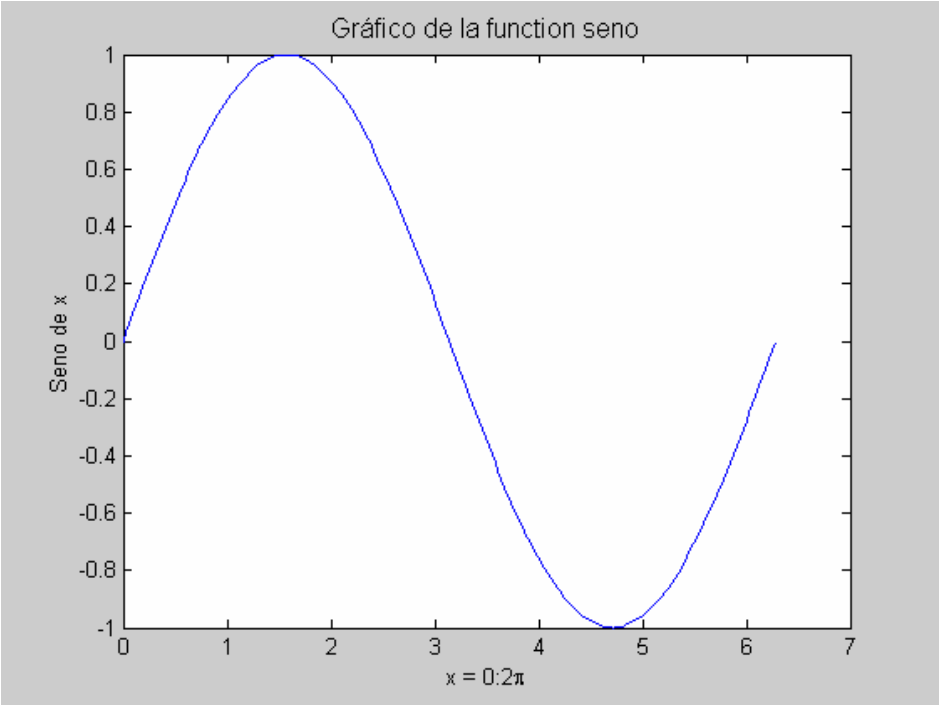
CREACIÓN DE UN GRÁFICO Para crear un gráfico bidimensional se dispone de la función ***plot***. La función plot tiene distintas formas, dependiendo de los argumentos de entrada. Si ***y*** es un vector, ***plot(y)*** produce un gráfico de segmentos de líneas uniendo los valores de los elementos de ***y*** versus el índice de los elementos de ***y***. Si se especifican dos vectores como argumentos, ***plot(x,y)*** produce un gráfico de y versus x.

Por ejemplo, las siguientes instrucciones utilizan el operador dos puntos (colon) para crear un vector de los valores de x variando desde cero hasta 2π, para calcular el seno de estos valores y graficar el resultado.

```
>>x = 0: pi/100 : 2*pi;
>>y = sin(x);
>>plot(x,y)
```

En el command window se pueden escribir a continuación las instrucciones para colocar un nombre a los ejes y adicionar un título. Los caracteres \pi crean el símbolo π.

```
>>xlabel('x= 0:2\pi')
>>ylabel('Seno de x')
>>title('Gráfico de la function seno','FontSize',12)
```



MÚLTIPLE JUEGO DE DATOS EN UN MISMO GRÁFICO

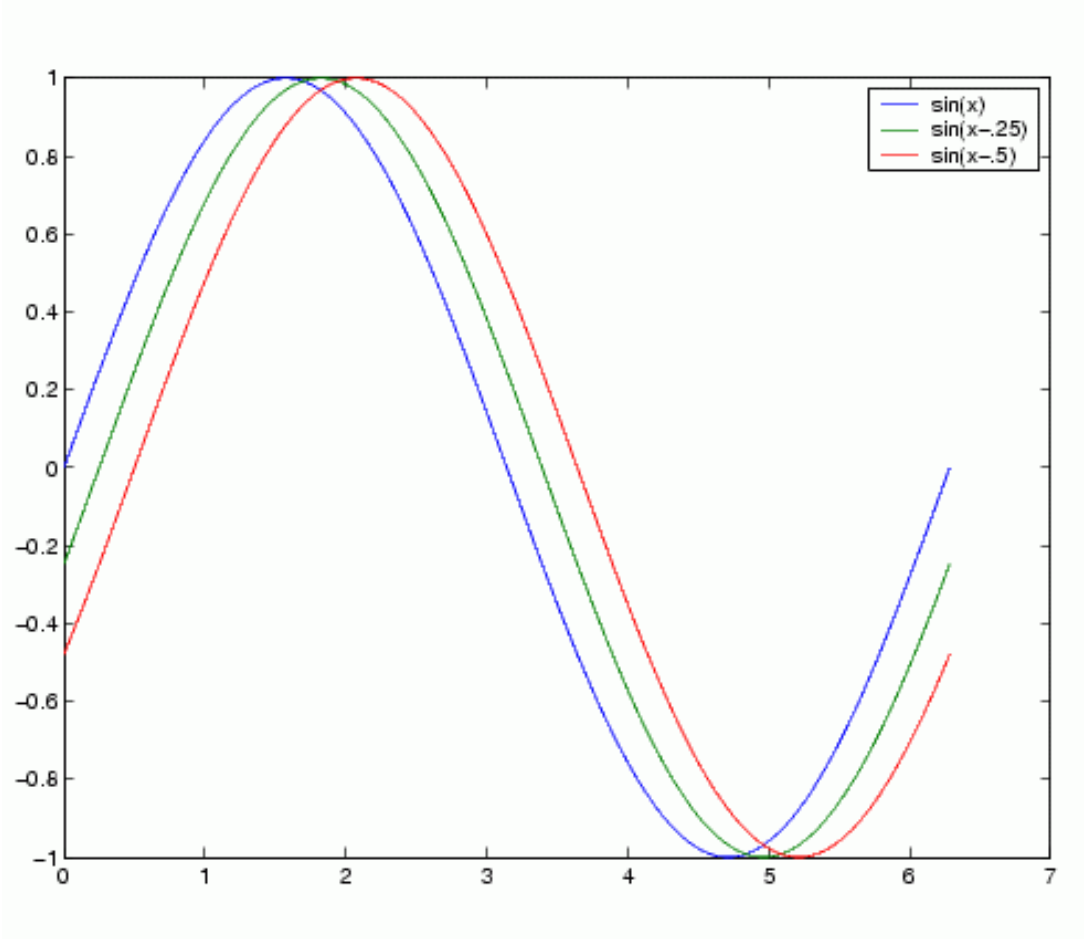
Múltiples pares de vectores x-y crean múltiples gráficos con una simple llamada a la sentencia plot. MATLAB cicla automáticamente a través de una lista de colores predefinida (pero configurable por el usuario) para permitir la discriminación entre cada conjunto de datos. Por ejemplo, estas instrucciones grafican tres funciones de x, cada una con un color separado que las distingue.

```
>>y2 = sin(x-.25);
>>y3 = sin(x-.5);
>>plot(x,y,x,y2,x,y3)
```

(véase que aquí el vector x es el mismo para las tres curvas, pero podría ser distinto, como x1,y x2,y2 x3, y3)

El comando legend provee una forma clara de identificar los gráficos individuales.

```
legend('sin(x)','sin(x-.25)','sin(x-.5)')
```



Matlab otorga colores por defecto: a la primera curva representada, x,y, se la dibuja en azul; a la segunda, verde y a la tercera, roja.

ESPECIFICANDO ESTILOS DE LÍNEAS Y COLORES

Es posible, sin embargo, especificar color, estilos de línea, y símbolos cuando represente sus datos usando el comando plot: ***plot(x,y,'marcador de estilo de color').*** Las cualidades del gráfico se pueden generar mediante instrucciones dadas desde el command window o desde el editor de archivos *.m previamente a la generación del gráfico (lo que siempre es más conveniente para automatizar) o desde la interfase de usuario, una vez terminado el gráfico. Se describirá a continuación como se escriben las instrucciones programando "a priori" el aspecto del gráfico.

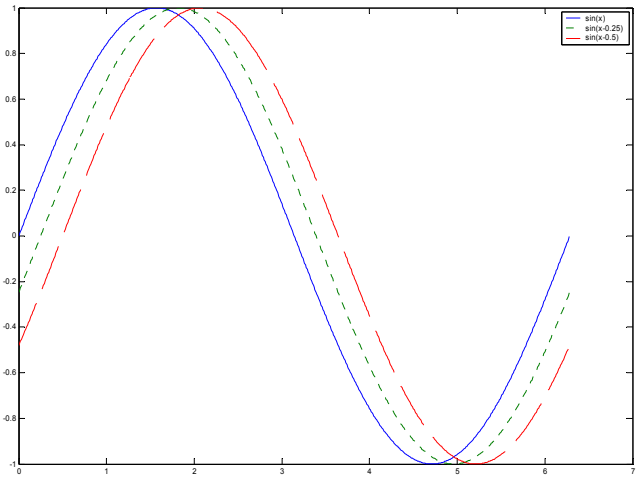
El marcador de estilo de color es una cadena que contiene de uno a cuatro caracteres (encerrados entre apóstrofes) construídos de un color, un estilo de línea, un tipo de marcador:

Símbolo	Color	Símbolo	Marcadores (markers)
y	yellow	.	puntos
m	magenta	o	círculos
c	cyan	x	marcas en x
r	red	+	marcas en +
g	green	*	marcas en *
b	blue	s	marcas cuadradas (square)
w	white	d	marcas en diamante (diamond)
k	black	^	triángulo apuntando arriba
		v	triángulo apuntando abajo
Símbolo	**Estilo de línea**	>	triángulo apuntando a la dcha
-	líneas continuas	<	triángulo apuntando a la izda
:	líneas a puntos	p	estrella de 5 puntas
-.	líneas a barra-punto	h	estrella se seis puntas
--	líneas a trazos		

LÍNEAS DE GRAFICACIÓN Y SIMBOLOS

Líneas En el caso de desearse una graficación destinada a ser observada en blanco y negro, como por ejemplo en fotocopias, conviene trazar líneas distintas en lugar de un tipo de línea y distintos colores. Por ejemplo, si la sentencia de graficación es

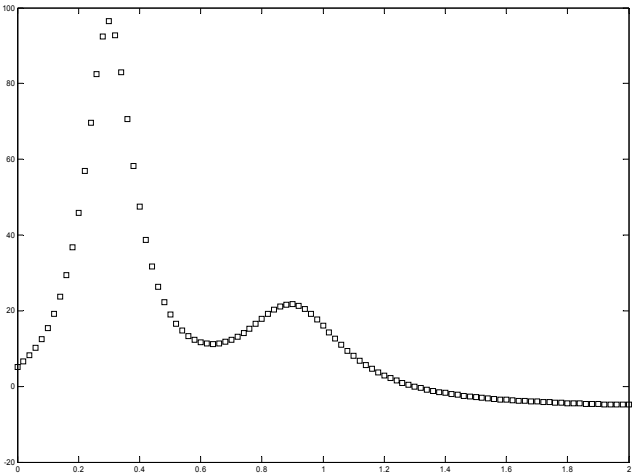
```
>>y = sin(x)
>>y1 = sin(x-0.25)
>>y2 = sin(x-0.5)
>>plot (x,y,x,y1,':', x,y2,'--')
```



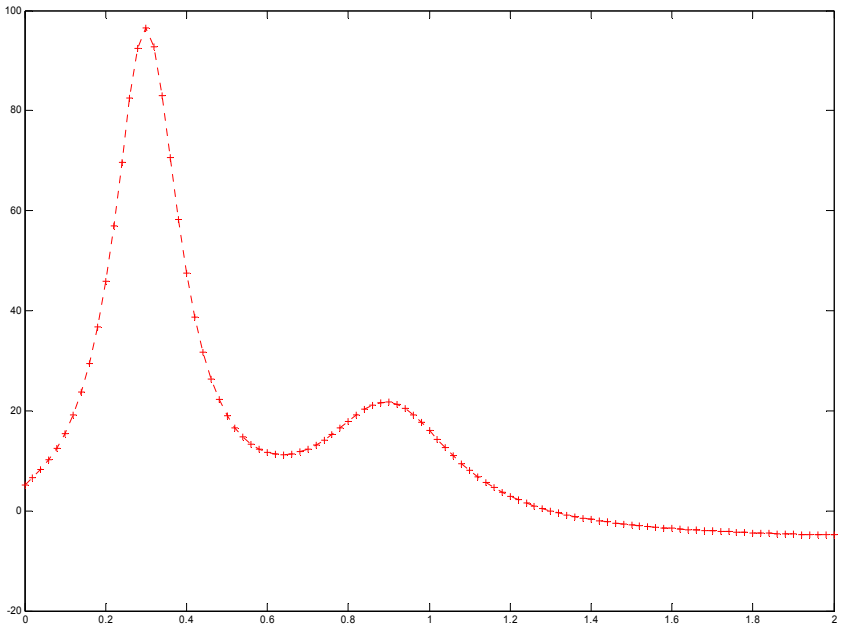
Símbolos Si se especifica un tipo de símbolo pero no un estilo de línea, MATLAB exhibirá sólo el primero. Por ejemplo,

```
clear
>>x=0:0.02:2;
>>y = 1 ./ ((x-.3).^2 + .01) + 1 ./ ((x-.9).^2 + .04) - 6;
>>plot(x,y,'ks')
```

Obsérvese que la instrucción ***plot(x,y,'ks')*** grafica cuadrados negros en cada punto de datos, sin conectar los marcadores con una línea.



Si deseamos que aparezcan líneas con símbolos, se puede escribir, por ejemplo ***plot(x,y,'r: +')*** que grafica una línea roja de puntos y coloca marcadores de signo más en cada punto.



Comandos *hold on*, *hold off*

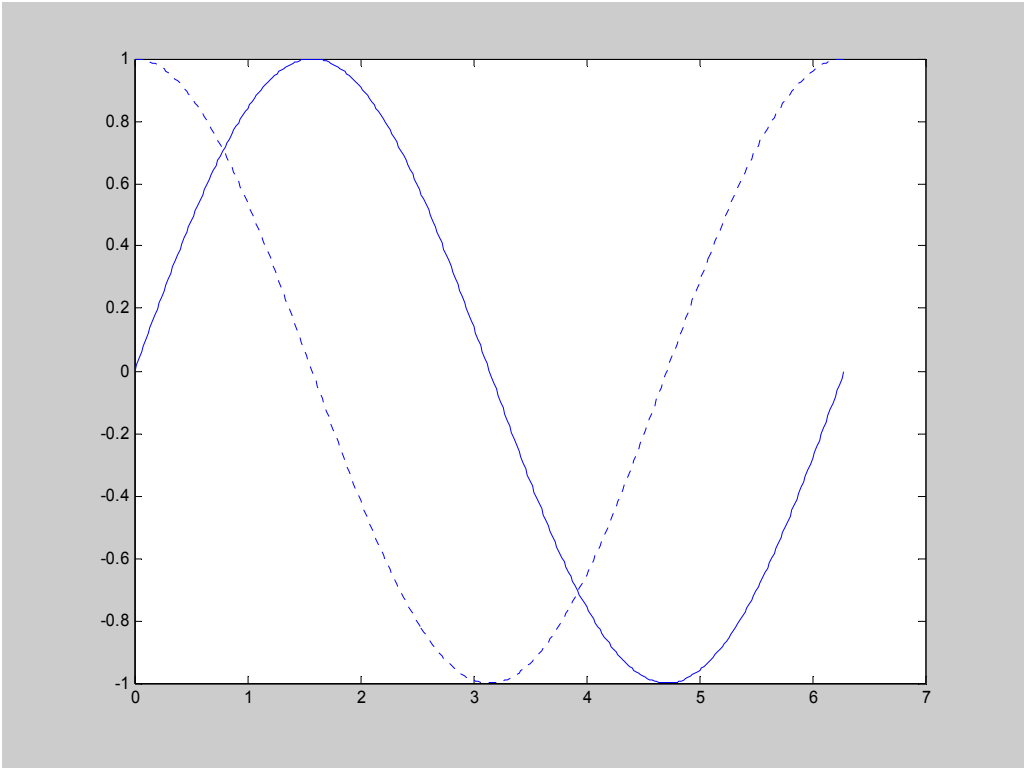
Normalmente, si se ejecuta una sentencia plot, y luego otra, en el curso de una sesión o programa, la ultima es la única que puede verse pues cada sentencia plot borra las anteriores. Por ejemplo

```
>>alfa = 0:0.02:2*pi;
>>y = sin(alfa);
>>plot (x,y)
>>z = cos(alfa)
>>plot(x,z)
```

permitiría visualizar sólo el gráfico del coseno, pero no el del seno. Una opción sería, como ya se vió, graficar conjuntamente las dos funciones, pero por distintas razones, surge a veces la necesidad de realizar una construcción secuencial de un gráfico para adicionar curvas gradualmente. Esto se puede hacer mediante el uso de la sentencia "hold on" (sostener o mantener activo):

```
>>x=0:0.02:2*pi;
>>y = sin(x)
>>plot(x,y)
>>hold on
>>z = cos(x)
>>plot(x,z,':')
>>hold off
```

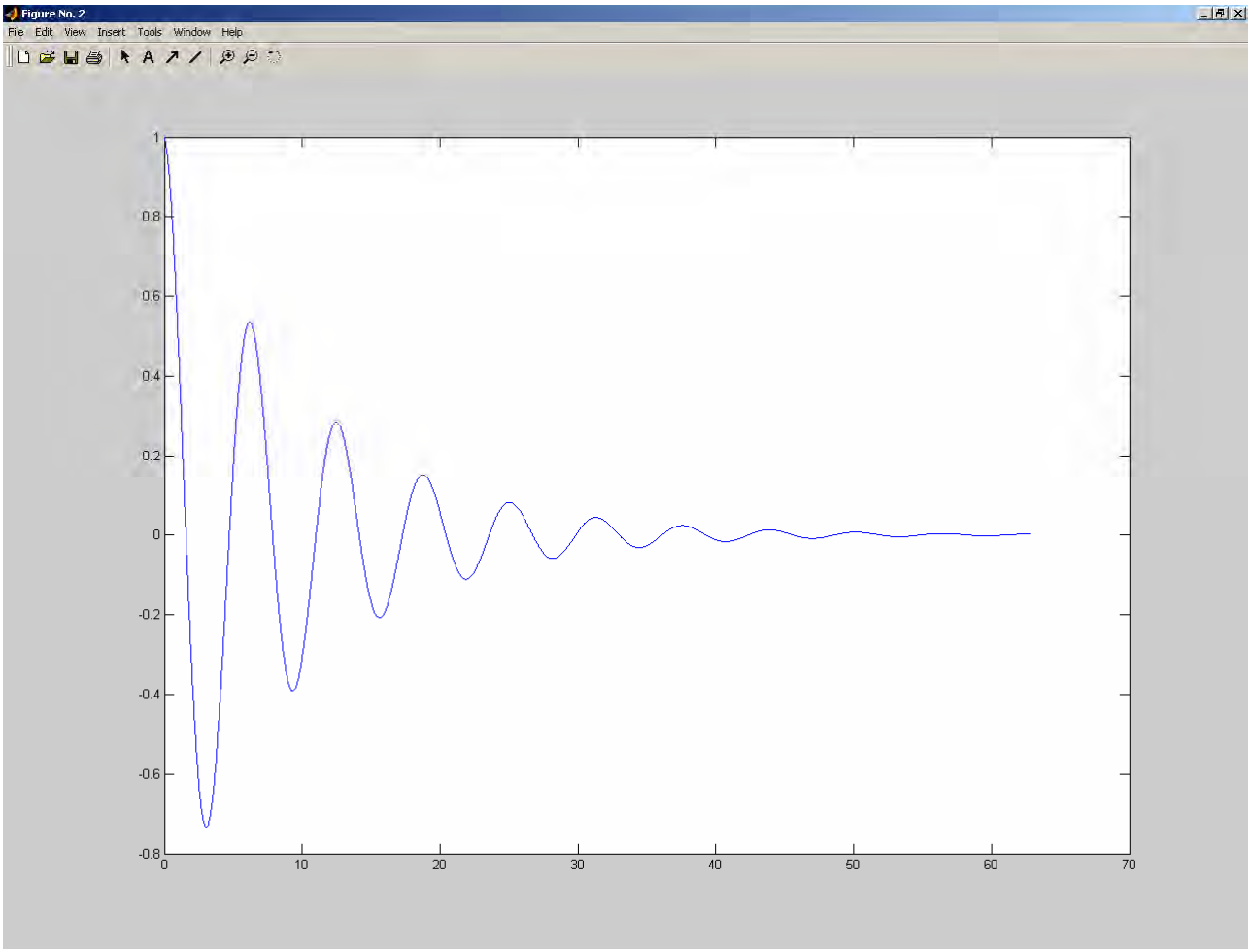
Aquí la graficación del coseno se personalizó en línea de puntos debido a que la nueva sentencia plot en realidad produce otro gráfico pero "transparente", que se superpone con el anterior, mantienendo las propiedades de la ventana de gráficos del anterior, pero no la memoria de las curvas, de manera que el coseno se graficaría en línea continua azul otra vez como una primera curva. El comando hold on permite entonces la superposición de varios gráficos que en la práctica se ven como uno. La configuración por defecto (cierre del gráfico) se reestablece utilizando la instrucción "hold off".



Comando figure (n)

El comando figure permite mantener simultáneamente varias figuras para su observación, lo que resulta sumamente útil para evaluar los resultados de un programa. Por ejemplo:

```
>>x = 0:0.02:2*pi;
>>y = cos(x)
>>z = exp(-x).*cos(x);
>>plot(x,y)
>>figure(2)
>>plot(x,z)
```



Si bien directamente se observa la figura 2, minimizando ésta se puede observar “detrás” la figura 1. El procedimiento es válido para exhibir numerosas figuras que se solapan como resultado de la ejecución de un programa.

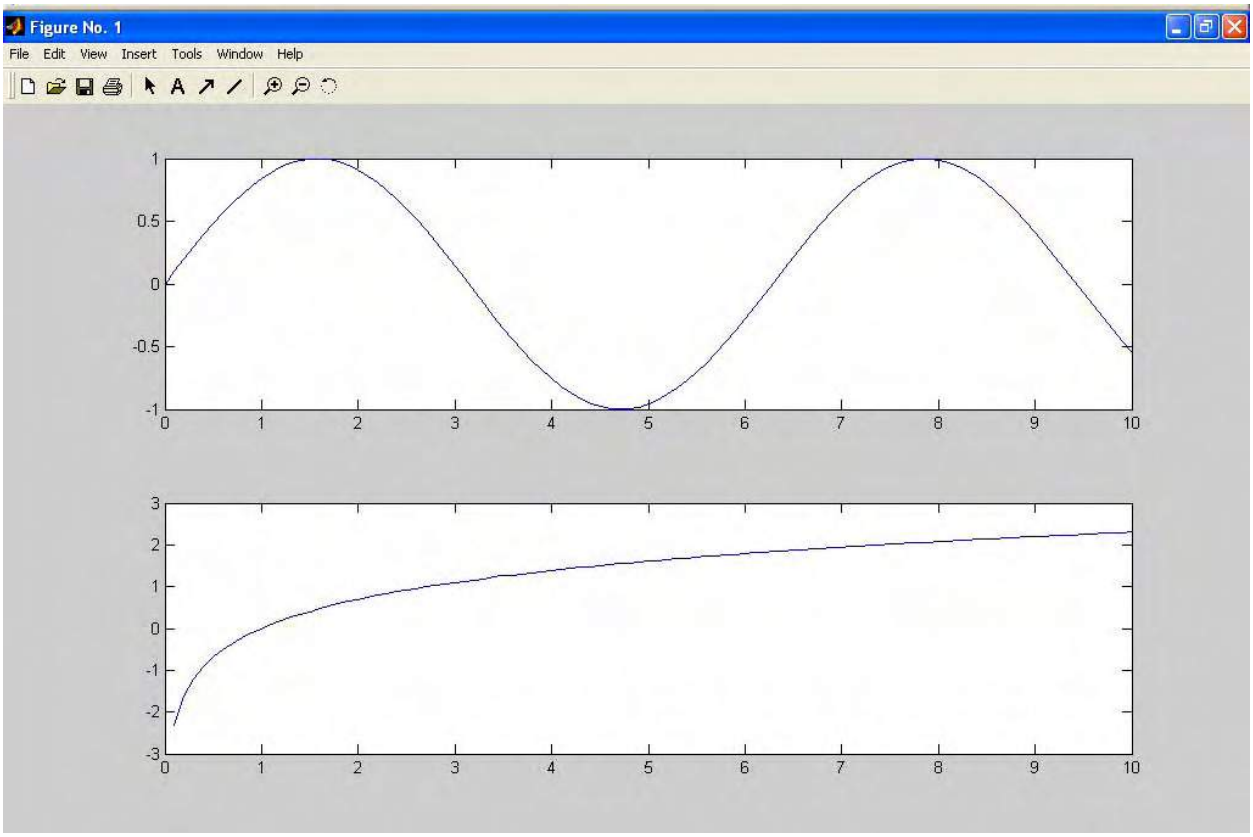
SUBDIVISIÓN DE LA VENTANA DE GRÁFICOS

A efectos de exhibir simultaneamente varias figuras sin solapamientos, se puede subdividir la ventana gráfica con la función ***subplot()*** en m particiones horizontales y n verticales. Cada una de esas particiones se puede llamar "subgráfico". El subíndice "i" identifica la subdivisión que se convierte en activa.

Forma general: subplot(m,n,i)

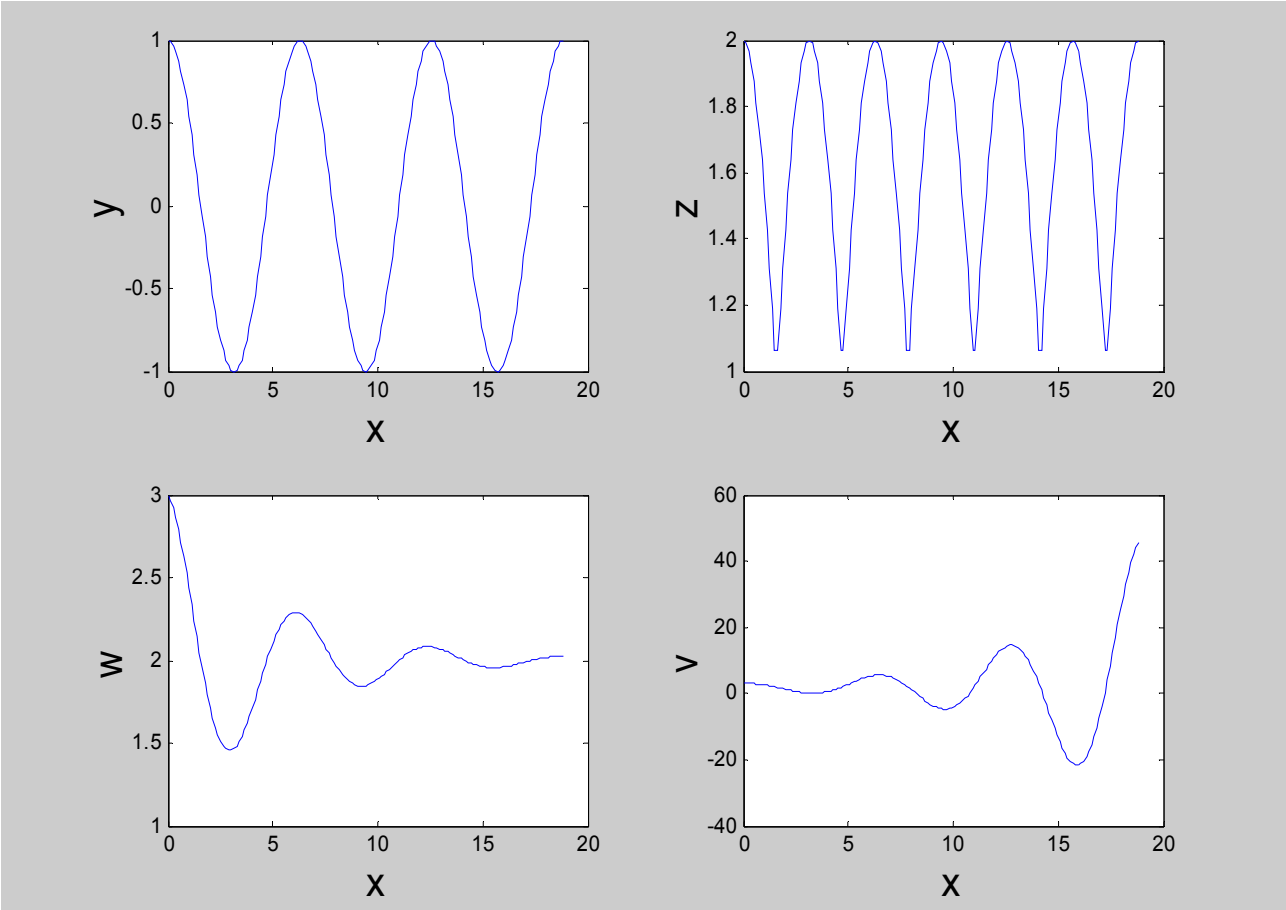
Ejemplo de subdivision en dos filas y una columna:

>>x=0:0.1:10; subplot(2,1,1), plot(x,sin(x)); subplot (2,1,2), plot(x,log(x))
Warning: Log of zero.



Ejemplo de subdivision en dos filas y dos columnas

```
x=0: pi/25: 6*pi;
y= cos(x);
z= abs(cos(x))+1;
w=exp(-0.2.*x).*y+ 2;
v= exp(+0.2.*x).*y +2;
subplot(2,2,1), plot(x,y); xlabel ('x', 'Fontsize', 18);ylabel('y','Fontsize', 18)
subplot(2,2,2), plot(x,z); xlabel ('x', 'Fontsize', 18);ylabel('z', 'Fontsize', 18)
subplot(2,2,3), plot(x,w);xlabel ('x', 'Fontsize', 18');ylabel('w','Fontsize', 18)
subplot(2,2,4), plot(x,v); xlabel ('x', 'Fontsize', 18);ylabel('v', 'Fontsize', 18)
```



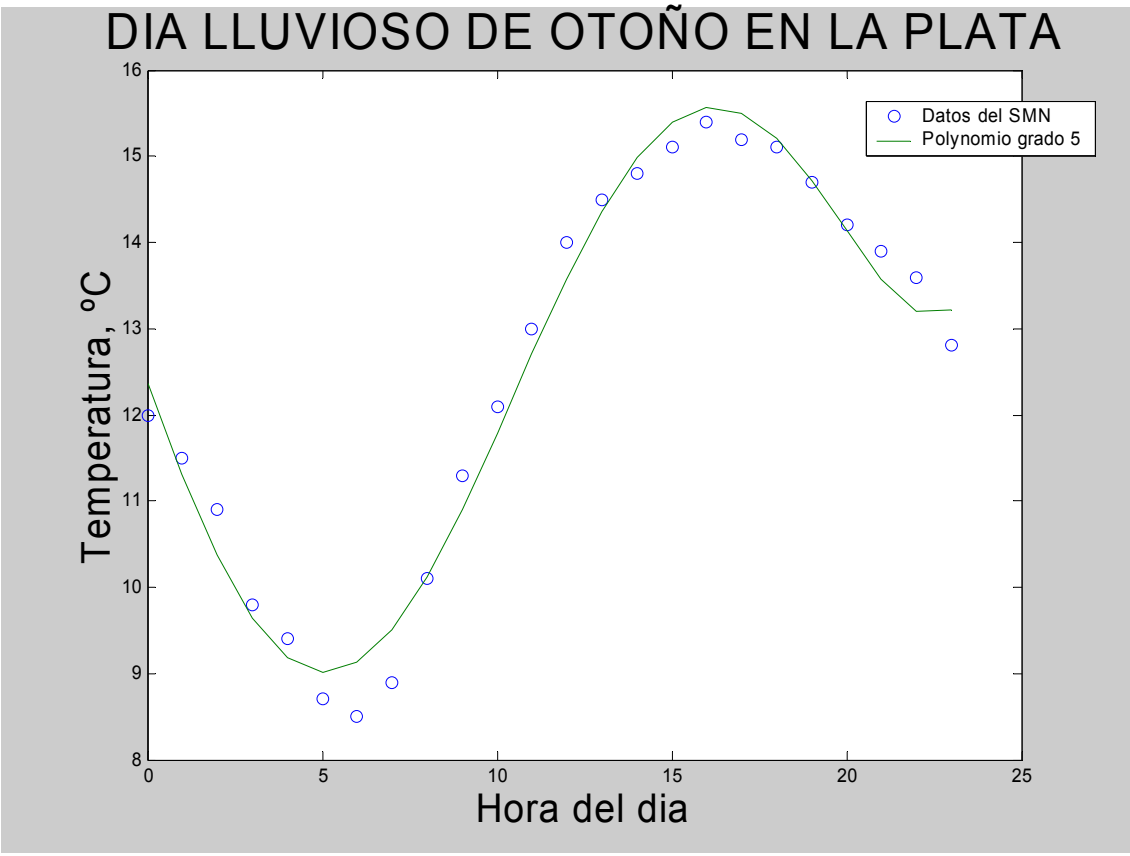
Como se advierte en la figura, las subventanas de gráficos se “llenan” por filas, primero y, luego z, luego w, y finalmente, v.

Graficación conjunta de observaciones y predicciones

Es muy frecuente en ingeniería comparar las observaciones medidas (por ejemplo la temperatura ambiente) con los cálculos realizados mediante un modelo matemático (sea éste meramente una función algebraica ajustada previamente a los datos, o un modelo teórico que interprete la física del sistema). El siguiente programa ilustra un ejercicio, donde se comparan datos de un día lluvioso de abril en La Plata, con un polinomio de grado 5 ajustado a ellos. Incidentalmente se muestran las útiles funciones preprogramadas de Matlab, "polyfit" que devuelve el vector de coeficientes del polinomio ajustado según el método de los cuadrados mínimos y "polyval"que calcula los valores del polinomio dado el vector de coeficientes y el vector de la variable independiente, t.

```
clear
t = 0:1:23;
T = [12 11.5 10.9 9.8 9.4 8.7 8.5 8.9 10.1 11.3 12.1 13 14 14.5 14.8 15.1 15.4 15.2...
      15.1 14.7 14.2 13.9 13.6 12.8 ];
p = polyfit (t, T,5)
Tp = polyval(p,t)
plot (t,T, 'o', t, Tp, 'Linewidth', 1);
Title ('DIA LLUVIOSO DE OTOÑO EN LA PLATA', 'Fontsize', 25)
xlabel ('Hora del dia', 'Fontsize', 20)
ylabel ('Temperatura, ºC', 'Fontsize', 20)
legend ('Datos del SMN', 'Polynomio grado 5')
```

Los datos de T, están indicadas por símbolos, mientras que los de Tp, que son cálculados, se los representa mediante el modo automático de Matlab (puntos unidos con líneas rectas, que se ven como una curva)



GRÁFICOS TRIDIMENSIONALES

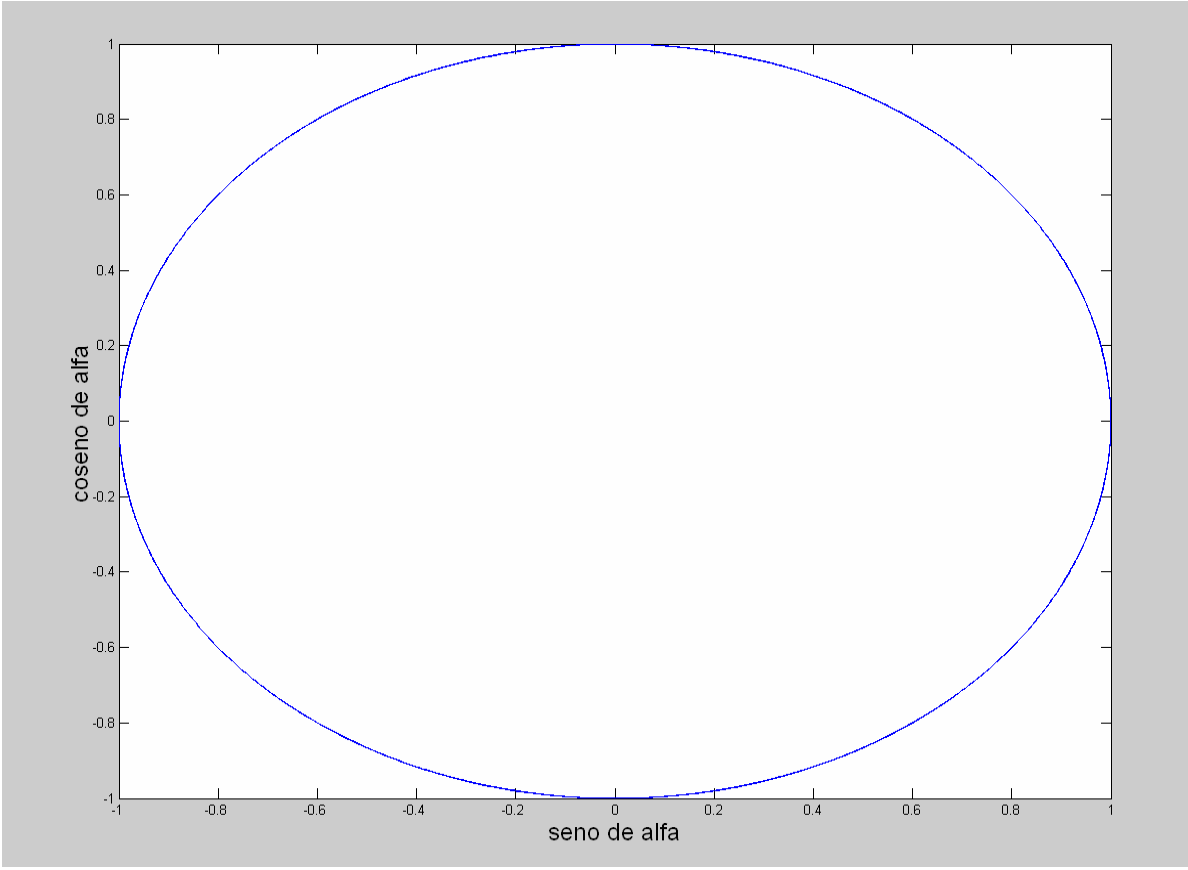
Gráficos de línea

La función ***plot3(x,y,z)*** es el análogo tridimensional de la función plot,. Permite dibujar puntos en un espacio 3-D cuyas coordenadas están contenidas en tres vectores fila (de 1 x n). Por defecto, plot3 une los puntos con líneas continuas al igual que plot, pero se puede configurar para que represente solamente los símbolos. Se puede utilizar para gráficos paramétricos.

Ejemplo

Si se grafica cos(α) en función de sen(α) por medio de la sentencia plot 2D ya estudiada tendríamos

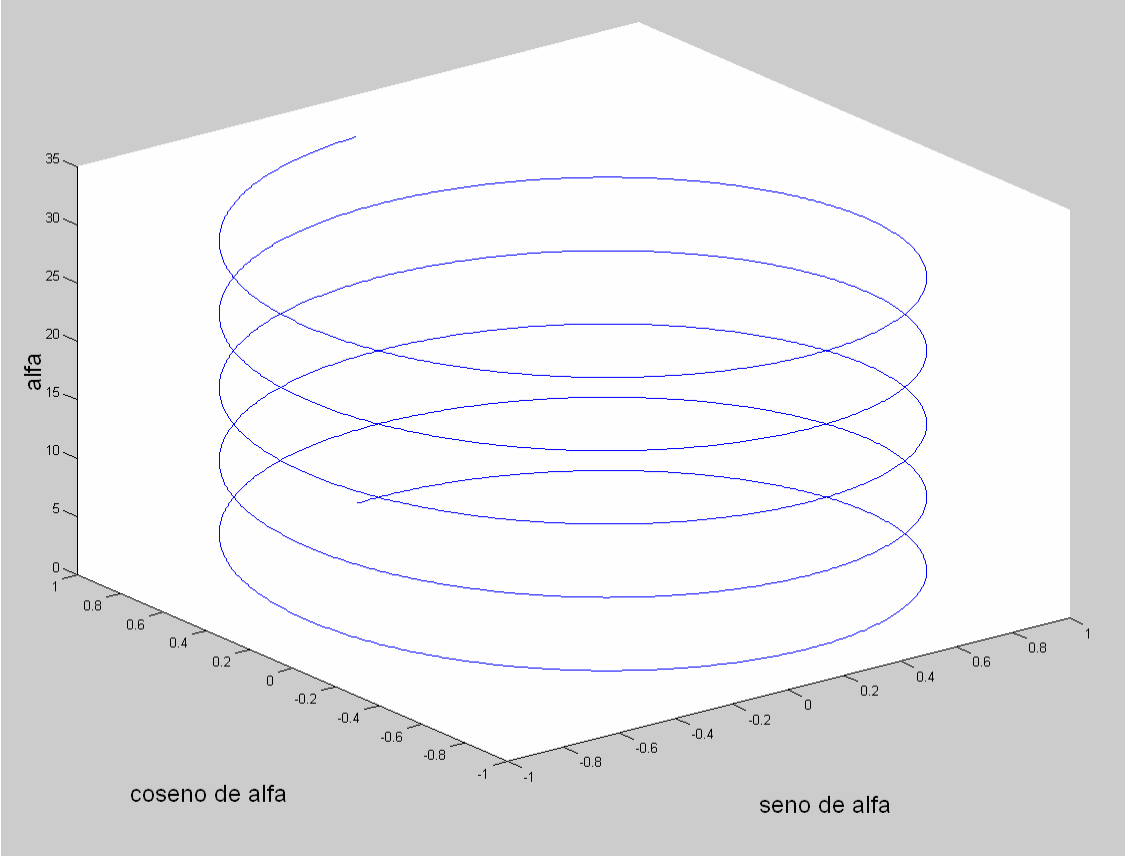
```
>> alfa = 0:0.01:10*pi;
>> y = sin(alfa);
>> z = cos(alfa);
>> plot(y,z); xlabel ('coseno de alfa', 'Fontsize', 18); ylabel ('coseno de alfa', 'Fontsize', 18);
>> plot(y,z); xlabel ('seno de alfa', 'Fontsize', 18); ylabel ('coseno de alfa', 'Fontsize', 18);
```



El gráfico, si tuviera una relación de aspecto 1:1 produciría una circunferencia por cada 2π de manera tal que la figura es en realidad la superposición de 5 circunferencias

Si ubicáramos el vector alfa en el eje vertical, las circunferencias irían ascendiendo, dando lugar a un gráfico tridimensional "tipo resorte".

```
>plot3(y,z,alfa, 'LineWidth',1);xlabel ('seno de alfa', 'Fontsize', 18); (continúa debajo)
 ylabel ('coseno de alfa', 'Fontsize', 18);zlabel ('alfa', 'Fontsize', 18)
```



SUPERFICIES

En los gráficos de superficie, se pueden representar valores de z como funciones de x e y. No obstante, la diferencia con el plot3 radica en que los vectores x e y deben ser transformados en matrices X e Y, por medio de la función "meshgrid".

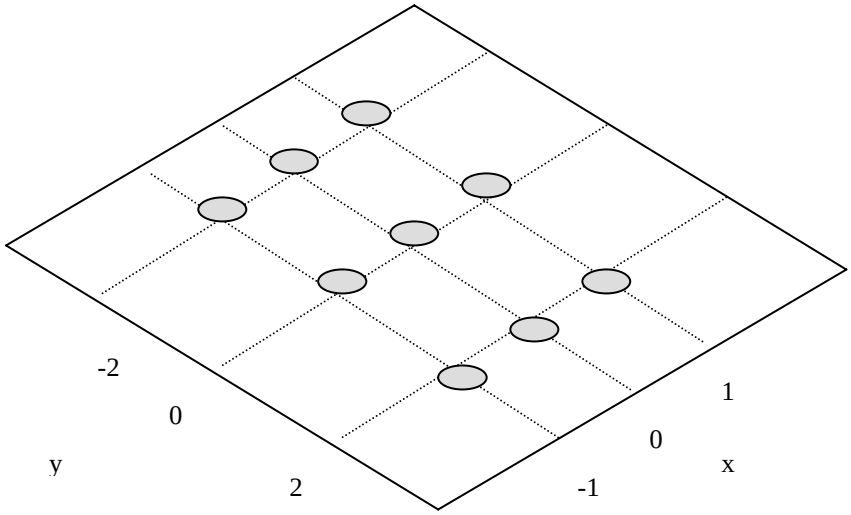
[X,Y]= meshgrid (x,y)

Por ejemplo, si x = -1:1; e y = -2:2:2

```
>> x = [-1,0,1]

x =
   -1    0    1

>> y = [-2,0,2]
y =
   -2    0    2
```

```
>> [X,Y]= meshgrid (x,y)
X =
   -1    0    1
   -1    0    1
   -1    0    1
Y =
   -2   -2   -2
    0    0    0
    2    2    2
```

Es decir, La matriz X contiene 3 filas iguales , cuyos valores son los del vector original x. El hecho de que sean 3 filas corresponde a que el vector original "y" tiene 3 valores.
La matriz Y, en cambio, contiene 3 columnas iguales, cada una de ellas con los valores originales del vector "y". En este caso, el hecho de que sean 3 columnas corresponde a que el vector original "x" tiene 3 valores.
Las matrices X e Y conforman los puntos del dominio, en el plano x-y, donde se van a calcular los valores de la función Z = f (X,Y)

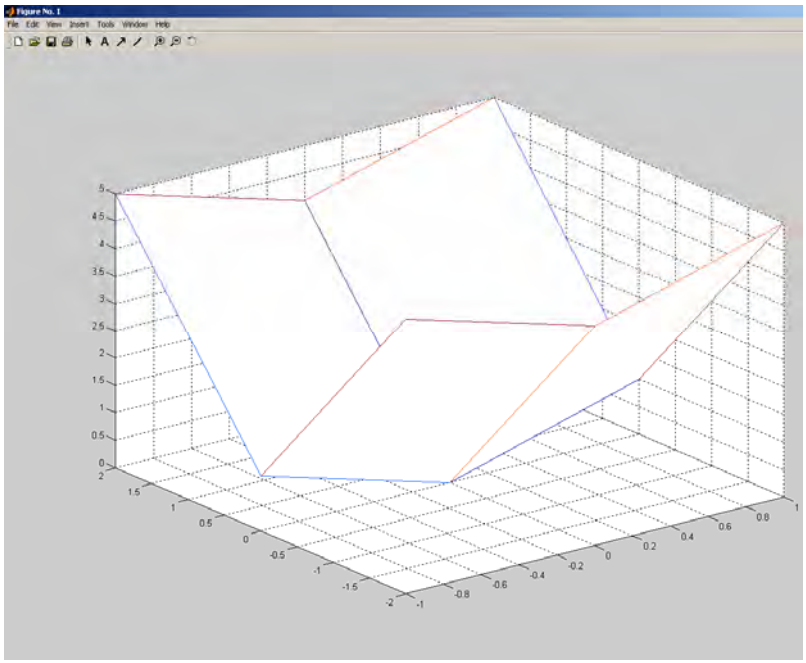


si por ejemplo, se calculan los valores de la matriz Z

```
>> Z = X.^2 + Y.^2
```

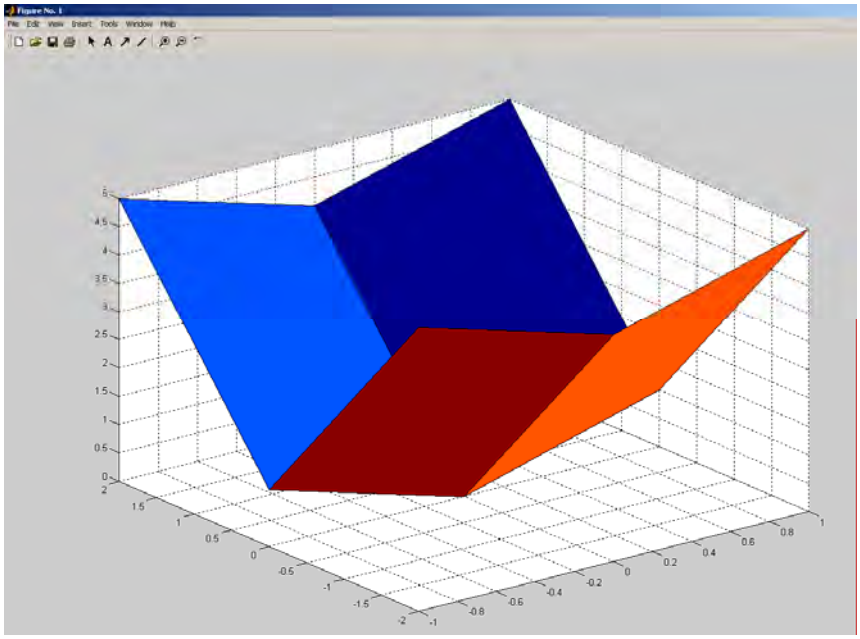
y luego se representan en 3 dimensiones como una superficie, usando la función "mesh"

>>mesh (X,Y,Z)
se obtiene



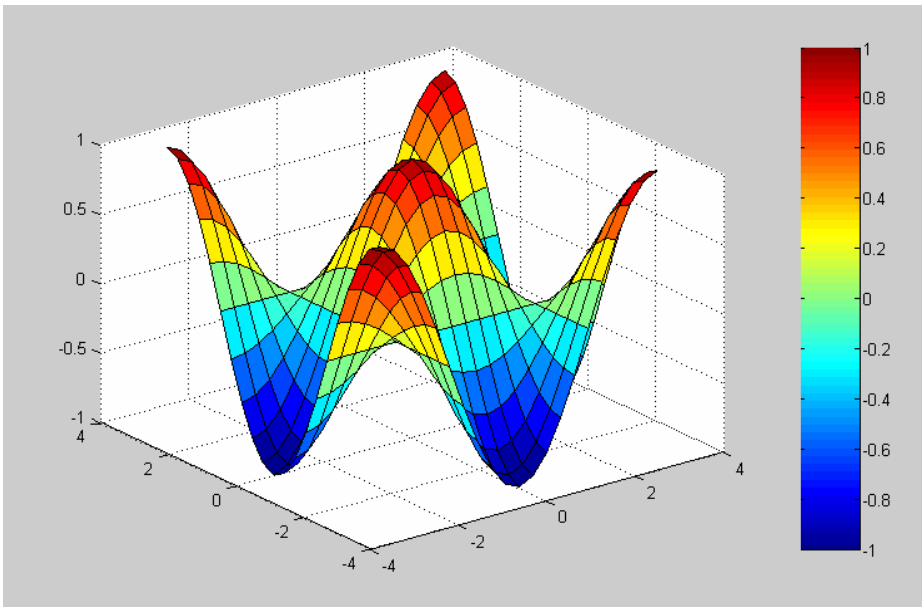
Así como la función mesh grafica la retícula en 3 dimensiones, uniendo los puntos de Z con líneas, la función "surf" colorea la forma 3D siguiendo un código de colores aclarado en colorbar. La función surf puede usarse directamente, luego de "meshgrid", no se necesita ejecutar la función "mesh" primero.

>> surf(X,Y,Z)



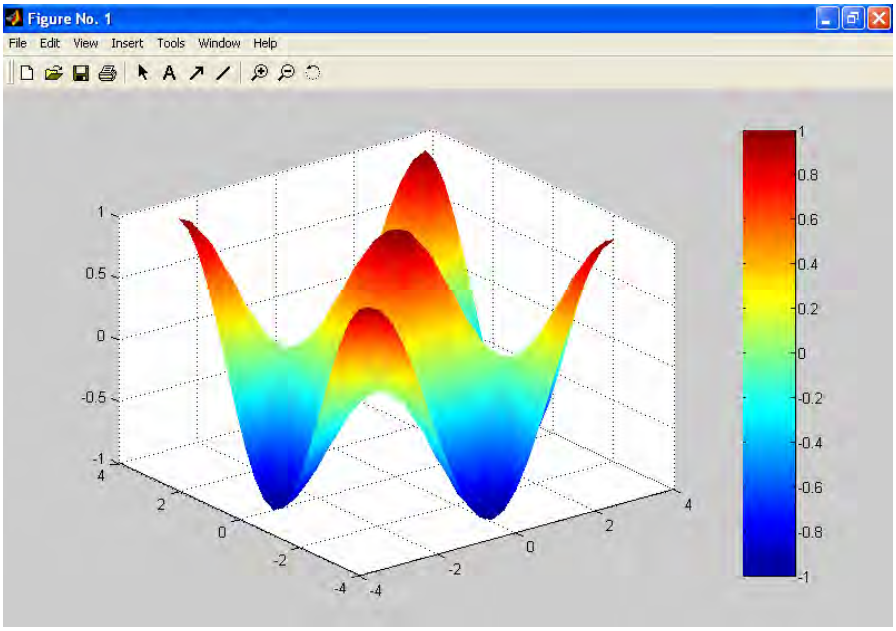
La superficies se pueden observar de varias maneras. Para controlar la dirección de observación se emplea función ***view(az, el)***. Los ángulos se definen en grados. Por defecto: **az** = -37.5, **el** = 30. Se verá otro ejemplo de mayor complejidad

```
>>[X,Y] = meshgrid (-pi:pi/10:pi, -pi:pi/10:pi);
>>Z = cos (X).* cos(Y)
>>surf(X,Y,Z)
>>view (30, 60)
>>colorbar
```



Las interfases entre zonas de distintos colores pueden suavizarse haciendo

>>shading interp



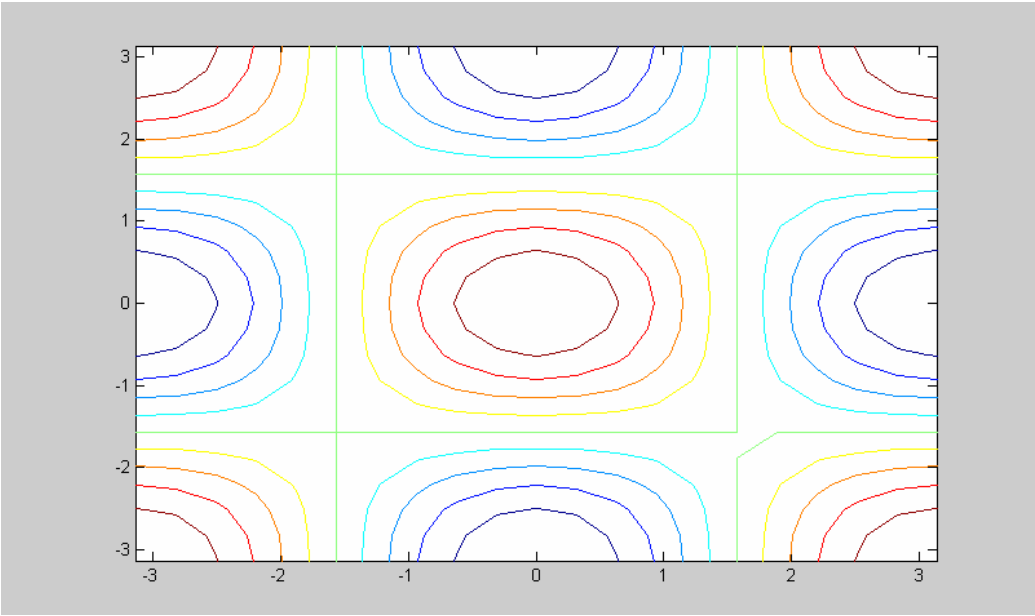
Las características particulares de este gráfico se pueden cambiar también desde la ventana de la Figura (Edit-Figure properties-Axes properties-Current object properties).

GRÁFICOS DE CONTORNO O CURVAS DE NIVEL

Los gráficos de contorno se utilizan para visualizar un gráfico de 3D en un sistema de coordenadas 2D. Un contorno se produce la superficie 3D es intersectada por un plano paralelo al X Y, a una dada altura Z. La repetición de este procedimiento a distintas alturas Z permite proyectar las curvas de nivel o gráficos de contorno sobre el plano XY de base. Siguiendo el ejemplo anterior, si se escribe

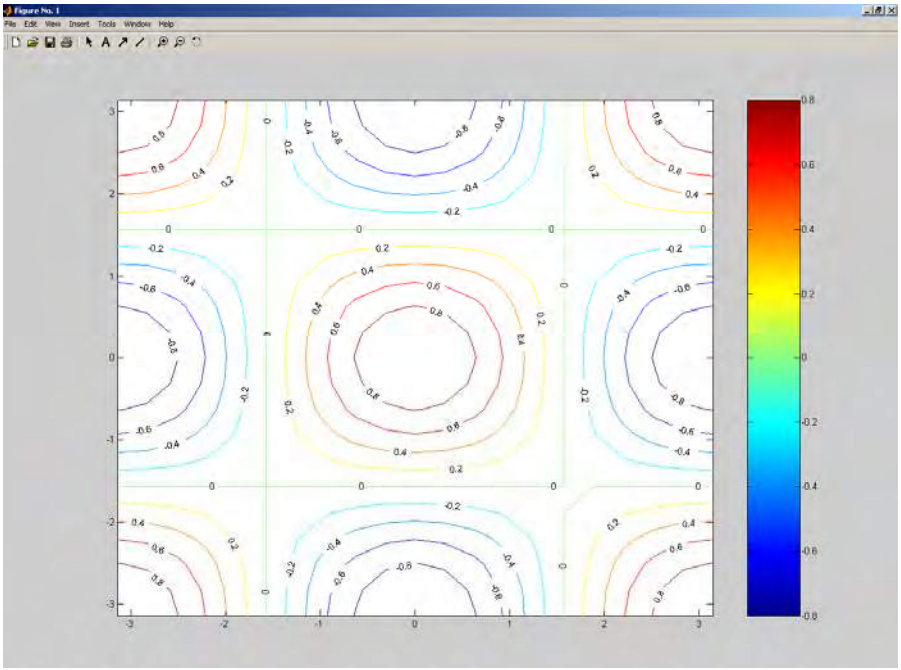
>>Contour (X,Y,Z)

se obtiene



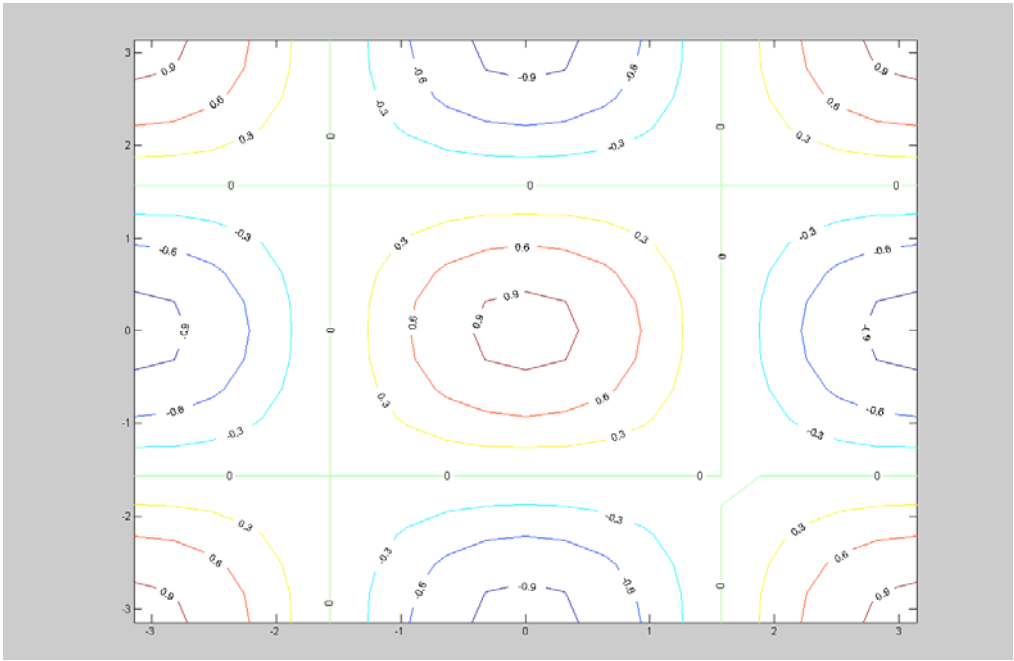
Pero este gráfico no permite observar a que valores de Z corresponden las curvas. Con este fin se guarda información del gráfico en las variables c y h, y luego se suministra ésta a "label" a efectos de que aparezcan los valores sobre las curvas. La sentencia "colorbar" permite asignar colores a las curvas de nivel, según sus valores de Z.

>> [c,h] = contour(X,Y,Z); clabel(c,h), colorbar



Es posible mostrar sólo los niveles requeridos de la variable Z, por ejemplo entre -0.9 y 0.9 variando de a 0.3, mediante

>>[c,h]= contour (X,Y,Z, [-0.9:0.3:0.9]);
>>clabel(c,h);



CAPITULO 5. PROGRAMACIÓN MATLAB PARTE I. COMPONENTES BÁSICOS DEL LENGUAJE.

Introducción. Parte general

El "command window" de Matlab opera con símbolo, el "prompt" >>, que "espera instrucciones" de la línea de comandos. En esa línea se pueden ejecutar instrucciones del usuario, funciones preprogramadas, introducir datos, y visualizar resultados. Estos cálculos utilizan la sintaxis y reglas de formación de expresiones del Lenguaje Matlab y conforman lo que podría denominarse "operación en "modo interactivo" (instrucción –enter-instrucción) comparable a la utilización de una moderna calculadora gráfica. Hasta ahora nos hemos concentrado en este aspecto interactivo del entorno Matlab.

Matlab es además, un lenguaje de programación de muy alto nivel. Los lenguajes de programación de mayor dificultad de uso, pero que permiten sacar máximo provecho de las posibilidades de los sistemas de cómputo se denominan de "bajo nivel", como el lenguaje binario; están también los de nivel intermedio como el Assembler (Ensamblador), que fueron empleados por programadores, especialmente durante los inicios de la computación digital (décadas de 1940-50). La familia de lenguajes "C" es de nivel intermedio-alto, muy utilizado en ingeniería electrónica para comandar la operación de dispositivos. Los lenguajes clásicos de alto nivel, que han sido usados por estudiantes de ingeniería química son Fortran (Formula Translator), Basic (Beginners all-purpose symbolic instruction code) y Pascal, entre otros. Son lenguajes donde el programador escribe casi todo el programa, puesto que las opciones de funciones preprogramadas no son muy variadas. El lenguaje Matlab, en cambio, facilita la utilización de las funciones que ya trae incorporadas en el entorno por lo que constituye realmente un "Centro de Cómputos" en una computadora personal. El programador Matlab, podrá seleccionar las partes a programar, y aquellos segmentos de programa donde utilizará recursos ya disponibles del sistema. Para eso, además de conocer las características del lenguaje, deberá conocer el entorno y el sistema de ayuda, a efectos de estar capacitado para emplear los recursos disponibles en forma eficiente.

En este curso se enseñará Matlab desde la óptica del programador clásico, como medio para aprender a programar. En otras asignaturas, y especialmente en la vida profesional, podrá utilizar el sistema en la forma que desee.

Archivos *.m

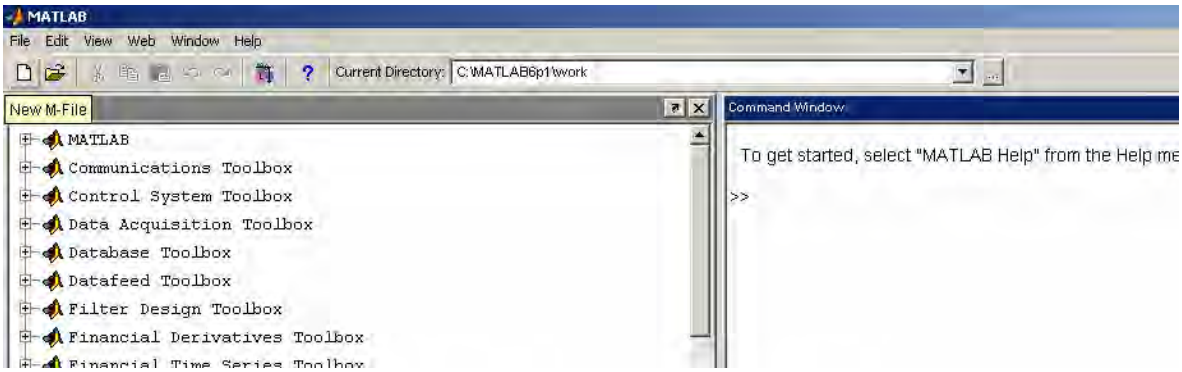
Cuando una serie de instrucciones dadas por el usuario se repite en Matlab puede ser agrupada y transformada en un programa propio del usuario (ya devenido programador). Se la copia al editor de archivos, se le da un ***nombredearchivo*.m** y se la ejecuta desde el command window invocando su nombre como si fuera una función más del entorno Matlab.

A menudo, los primeros programas suelen tener una estructura secuencial; no obstante, en otros casos, es necesario programar la toma de decisiones con la introducción de estructuras alternativas, y considerar la repetición de la ejecución de ciertos grupos de sentencias para distintos valores de determinadas variables, por medio de lazos repetitivos. Así es que surgen entonces los verdaderos programas Matlab (las "aplicaciones desarrolladas por el usuario"). Cualquier archivo ***.m** se ejecuta desde el command window, escribiendo su nombre, o desde otro archivo ***.m** que se considera "programa llamador", el que a su vez, se ejecuta desde el command window.

En otros lenguajes de alto nivel, el programa escrito en tal lenguaje se denomina "código fuente o programa fuente" y se traduce al lenguaje de máquina, antes de ejecutarlo. Esa traducción se denomina ***compilación.*** Los programas compilados pueden correrse para distintos juegos de datos, que se tipean en archivos para texto ASCII o "plano" (plain text) (como los del bloc de notas). En Matlab, el usuario no detecta la compilación puesto que ejecuta el código fuente directamente desde el command window. Un programador Matlab puede asignar todos los datos en un archivo *.m mediante asignaciones directas, y llamarlo desde otro archivo *.m, simplemente intercalando su nombre como si fuera una instrucción más del segundo programa. Así, se desdibujan los límites entre programa fuente y archivo de datos. Esta potente característica Matlab simplifica enormemente la entrada de datos. Si bien la sintaxis y las reglas de formación de expresiones se deben conocer ya para la utilización interactiva del entorno, vamos a estudiarlas con mayor atención en este capítulo.

EDITOR DE MATLAB, para la realización de programas, **nombredearchivo.m**

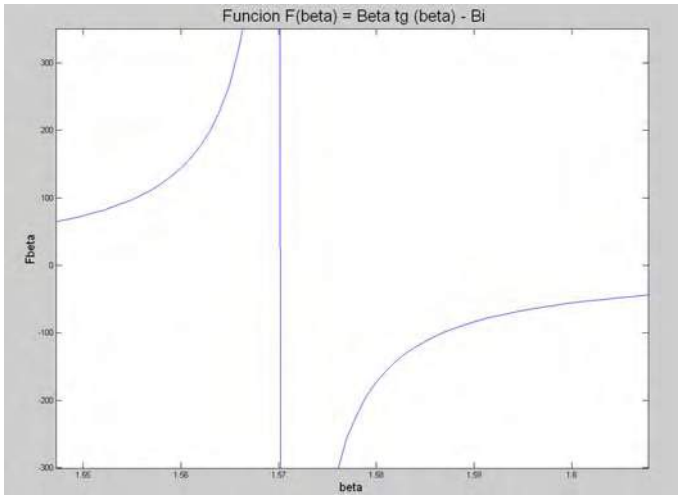
En el escritorio de Matlab, llevamos el puntero del ratón a la esquina superior izquierda, menú "File". Elegimos "New" para un programa nuevo (o botón con página en blanco), u "open" si queremos trabajar con un programa existente



Ejemplo de uso

Gráfica de una función F(β) = β tg β – Bi, que tiene gran interés en la resolución de ecuaciones diferenciales parciales de transferencia de calor y materia

El editor maneja texto plano, pero agrega color verde a los comentarios (precedidos por %) que son instrucciones que sirven para hacer más legible el programa, pero que no tienen efecto computacional alguno. Las cadenas de caracteres en ejes y títulos de gráficos se colorean automáticamente de un color rojizo. Para ejecutar o "correr" el programa, se usa el menú "debug" y allí se oprime "run", o bien se activa el. Una función muy útil del editor es en el menú desplegable "edit", es "find and replace", es decir, búsqueda y reemplazo. Es muy útil para corregir errores. La conocida función "cut o copy and paste" (cortar o copiar y pegar) es sumamente práctica a la hora de evitar retipear demasiado código.
La función del ejemplo tiene infinitas raíces. Aquí se amplió el gráfico para mostrar la primera, partiendo de β=0, que es de alrededor de 1,57.



Aspectos básicos del Lenguaje MATLAB

Tipos de datos

Como se ha dicho, cuando Matlab trabaja con datos numéricos, utiliza matrices, que son tipos de datos formados por elementos. Como caso particular de una matriz de 1x1 Matlab trabaja con escalares, y el lenguaje siempre opera en doble precisión, otorgando 8 bytes (64 bits) de memoria a cada elemento de la matriz. De los 64 bits, emplea 53 para la mantisa, lo que implica tratar 15 cifras decimales exactas y deja 11 para el exponente de 2. Matlab también puede trabajar con cadenas de caracteres, y con otros tipos de datos como matrices de más de dos dimensiones (hipermatrices), matrices dispersas, vectores, matrices de celdas, estructuras, clases y objetos.

Números Para números, MATLAB usa notación decimal convencional, con un punto decimal opcional y un signo opcional negativo o positivo. *La notación científica* utiliza la letra e para especificar un factor de escala de potencias de 10. Los *números imaginarios* usan tanto *i* como *j* como sufijo. Algunos ejemplos correctos de números se muestran en el recuadro.

```
3          -99        0.0001

9.6397238  1.60210e-20  6.02252e23 1i        -3.14159j     3e5i
```

Todos los números se almacenan internamente usando el formato largo estándar para operaciones de punto flotante. Los números de punto flotante, tienen una *precisión* finita de aproximadamente 16 dígitos significativos y un *rango* de aproximadamente 10^{-308} a 10^{+308}.

Números complejos Matlab permite ingresar números complejos en la notación matemática usual

```
>> 2 + 5 j
ans = 2.000 + 5.000 i
```

Si se coloca una expresión cuyo resultado es imaginario, Matlab lo visualiza como tal

```
a= sqrt(-2)
a =
      0 + 1.4142i
```

Matlab permite generar matrices de números complejos

```
>> A = [1+2i  2+3i;  -1+i  2-3i]

A =
 1.0000 + 2.0000i       2.0000 + 3.0000i
-1.0000 + 1.0000i       2.0000 -  3.0000i
```

Asimismo, Matlab dispone de la función COMPLEX que genera un número complejo a partir de sus argumentos reales e imaginarios

Ej:

```
>> complex (1, 2)
ans =
  1.0000 + 2.0000i
```

Como se ha visto cualquier cadena de caracteres puede asignarse a una variable, sin declaraciones previas ni dimensionamiento

```
>>m = ‘cadena de caracteres’
```

EXPRESIONES ARITMETICAS:

Como otros lenguajes de programación, MATLAB provee *expresiones aritmética*s, pero a diferencia de aquellos lenguajes, involucran matrices enteras. Las expresiones están compuestas de:

- Variables aritméticas
- Números o constantes
- Operadores
- Funciones

Variables o identificadores del usuario

Son cantidades que puede variar durante el desarrollo de un programa o durante una sesión del command window. Para nombrarlas, se pueden usar combinaciones de letras, números y el guión bajo, hasta 31 caracteres. Debe comenzar por una letra. **Matlab diferencia entre mayúsculas y minúsculas**, así que PianO y piaNo no son lo mismo.

Como norma inicial de ayuda nemotécnica, se sugiere nombrar las matrices con mayúsculas, y vectores o escalares con minúsculas. Las expresiones se escriben en una línea, similarmente a los otros lenguajes.

Por ejemplo, la expresión cinemática para el espacio recorrido por un objeto uniformemente acelerado

$$x_0 + v_0 t + \frac{1}{2} a t^2$$

se puede escribir **>> x0 + v0.*t + 0.5.*a.*t.^2**

que, de estar definidos *x0*, *v0*, *t*, y *a*, va a dar un resultado que Matlab, asigna a una variable *ans (por answer,* respuesta*)*

Si se dividiera esta expresión por el tiempo transcurrido se obtendría la velocidad promedio,

>> v = (x0 + v0*t + 0.5*a*t^2)./t

En este caso tenemos una "asignación" deliberada, a la variable (ver capítulo "diagramación"). A la derecha del signo igual, puede haber una expresión, a la izquierda sólo una variable. Si existiera un móvil cuya velocidad fuera en todo momento 1 m/s menor que la del previo, se podría hacer
>>u = v-1

Operadores aritméticos matriciales

Las expresiones usan los operadores aritméticos y reglas de precedencia habituales.

+	**Suma (adición)**
-	**Resta (Sustracción)**
*****	**Multiplicación**
/	**División**
****	**División izquierda (Operación del álgebra lineal)**
^	**Potenciación**
'	**Transposición conjugada compleja**
( )	**Especificación del orden de evaluación**

Operadores de arreglos. Cuando se las lleva fuera del álgebra lineal, las matrices se consideran arreglos numéricos bidimensionales y las operaciones aritméticas se realizan elemento a elemento. Esto implica que la suma y resta son iguales para arreglos y matrices, pero para operaciones multiplicativas es diferente. MATLAB usa un punto como parte de la notación para operaciones multiplicativas con arreglos. La lista de operadores incluye:

+	**Suma**
-	**Resta**
.*	**Multiplicación elemento por elemento**
./	**División elemento por elemento**
.	**División izquierda elemento por elemento**
.^	**Potenciación elemento por elemento**
.'	**Transposición de arreglos no conjugada**

Ejemplos de asignaciones Se han visto ya varios ejemplos de expresiones MATLAB. Aquí tenemos unos pocos ejemplos adicionales, y los valores resultantes.

```
Fuerza = masa .* aceleracion;
Fuerza_grav= k.*m1.*m2./d.^2;
hipot = sqrt (cat1.^2+cat2.^2)
intensidad = voltaje./resistencia
a = abs(3+4i)
a =
    5
```

(recuerde que calcula la raíz cuadrada de la suma de los cuadrados de los argumentos)

```
z = sqrt(besselk(3,1))
z =
    7.1013
```

Las expresiones pueden incluir también los valores que pueden accederse a través de subíndices.

```
b = sqrt(a(2)) + 2*c(1)

        b =
            7
```

c(2,1) = a(2,1) + b(2,1)

¿Cuando usar operadores multiplicativos de elemento a elemento o matriciales?

Con este fin, podemos hacer la tabla siguiente, considerando dos cantidades a y b

a	b	a x b	$\frac{a}{b}$	a^b
escalar (1 x 1)	escalar (1 x 1)	* y .* equivalentes	/ y ./ equivalentes	^ y .^ equivalentes
escalar (1 x 1)	Matriz o vector	* y .* equivalentes	sólo ./	sólo .^
Matriz o vector	Matriz o vector	* para algebra lineal y .* para operaciones elemento a elemento	/ para algebra lineal y ./ para operaciones elemento a elemento	sólo con.^
Matriz o vector	Escalar (1 x 1)	* ó .* da lo mismo	/ ó ./ dan lo mismo	^ y .^ dan distinto

Conclusión: dado que entre dos escalares, los operadores matriciales y elemento a elemento son equivalentes, y teniendo en cuenta que la mayoría de los arreglos se procesan elemento a elemento, se sugiere trabajar con operadores .*, ./ y .^, excepto cuando deliberadamente quiera trabajar en operaciones matriciales o del algebra lineal, tales como multiplicación de matrices.

Ingreso de largas líneas de comandos

Si una instrucción no cabe en una sola línea, o le parece que queda mal una expresión demasiado larga, predefina grupos de variables y/o utilice tres puntos, seguidos por Enter para indicar que la instrucción continúa en la línea siguiente. Por ejemplo, consideremos la expresión que permite calcular el área superficial de un elipsoide de ejes l_1, l_2 y l_3

$$A_{gp} = \frac{\pi}{2} l_1 \left(\frac{l2+l3}{2} \right) \left(\frac{l2+l3}{2l_1} + \frac{1}{\left(\frac{\sqrt{l_1^2 - \left(\frac{l2+l3}{2} \right)^2}}{l_1} \right)} \arcsin \left(\frac{\sqrt{l_1^2 - \left(\frac{l2+l3}{2} \right)^2}}{l_1} \right) \right)$$

La expresión Matlab quedaría:

l_1=10;l_2=3; l_3=2.5;

```
agp = (pi./2).*l1.*0.5.*(l2+l3)*(  0.5 .*(l2+l3)./l1+ 1./(sqrt ( l1.^2 - (0.5.*(l2+l3)).^2 )./l1   ).*asin ( sqrt (  l1.^2- (0.5.*(l2+l3) ).^2  ) ./l1  ))
```

se podría separar en dos líneas así

```
agp = (pi./2).*l1.*0.5.*(l2+l3)*(  0.5 .*(l2+l3)./l1+ ...
1./(sqrt ( l1.^2 - (0.5.*(l2+l3)).^2 )./l1   ).*asin ( sqrt (  l1.^2- (0.5.*(l2+l3) ).^2  ) ./l1  ))
```

O mejor, se podrían predefinir grupos de variables U, y l_m

$$U = \frac{\sqrt{l_1^2 - l_m^2}}{l_1} \quad \text{y} \quad l_m = \frac{l_2 + l_3}{2}$$

Así la expresión del área superficial geométrica quedaría:

$$A_{gp} = \frac{\pi}{2} l_1 l_m \left(\frac{l_m}{l_1} + \frac{1}{U} \arcsin U \right)$$

permitiendo escribir en Matlab una expresión de asignación mucho más compacta

```
agp = (pi./2).*l1.*lm.*( lm./l1 + (1./U).*asin(U));
```

Optimización del uso de paréntesis en expresiones aritméticas

Se va a escribir la ec. de Ergun, conocida en la ingeniería química, que calcula la pérdida de presión que experimenta un fluido al atravesar, con velocidad superficial V, un lecho relleno de altura o longitud L, caracterizado por una porosidad ε y un tamaño de partícula D_p. La viscosidad y densidad del fluido son μ y ρ, respectivametne

$$\frac{\Delta p}{L} = 150\frac{(1-\varepsilon)^2 \mu}{d_p^2 \varepsilon^3} V + 1{,}75\frac{\rho(1-\varepsilon)}{d_p \varepsilon^3} V^2$$

```
eps = 0.4; mu= 1.8e-5; dp=3e-3; rho= 1.2; v= 0.5;
Deltap_L= (150*(1-eps)^2 *mu) /((dp^2)*(eps^3))*v +(1.75* rho*(1-eps))/ (dp*(eps^3))*v^2
Deltap_L =
 2.4844e+003
```

Comentarios

1) Obsérvese que la expresión Matlab escrita tiene un número mayor de paréntesis que los necesarios. El ejercicio propuesto es minimizar el número de paréntesis.

2) Como crítica a la expresión escrita de Ergun en Matlab, se puede decir que es muy larga, lo que la hace más proclive a incorporar errores. Imagínense que hubiéramos tipeado 175 en lugar de 1,75. Matlab se hubiera dado cuenta?. Y si hubiéramos omitido escribir μ (es decir, mu)?.

Escriba la ecuación de una forma más compacta y minimice el número de paréntesis.

Ejercicio escriba en Matlab la ecuación de GAB (Guggenheim, Anderson, De Boer) de la adsorción

$$W = \frac{W_m C k a}{(1-k a)(1-k a + C k a)}$$

donde W es la cantidad de adsorbato depositado, por unidad de masa de adsorbente, y "a" es la actividad del adsorbato en la atmósfera que rodea al adsorbente. C, k y Wm son coeficientes de la ecuación, que deben determinarse para cada par adsorbente-adsorbato utilizando datos experimentales. Haga un ejemplo en el command window, calculando W vs a, para a variando entre 0,01 y 0,99. Utilice los coeficientes siguientes: W_m = 0,07; C = 20 y k = 0.94. Represéntelos mediante la sentencia "plot".

Edición de la línea de comandos

Las teclas de flecha o control en su teclado permiten recordar, editar y reutilizar comandos tipeados con anterioridad. Por ejemplo, suponga que, equivocadamente, Ud. ingresa

u = squrt (l1.^2 – lm.^2);
habiendo escrito mal *sqrt*. MATLAB responde con *Undefined function or variable 'squrt'*.
En lugar de retipear la línea entera, simplemente presione la tecla ↑. El comando equivocado se muestra nuevamente. Use la tecla ← para mover el cursor hacia la zona a corregir. El uso repetido de la flecha ↑ "recuerda" líneas previas. Tipeando unos pocos caracteres, y luego la tecla ↑ encuentra una línea previa que comienza con esos caracteres. Asimismo, se pueden copiar comandos previamente ejecutados del historial de comandos (Command History). La lista de las distintas teclas disponibles para la edición de líneas de comando es

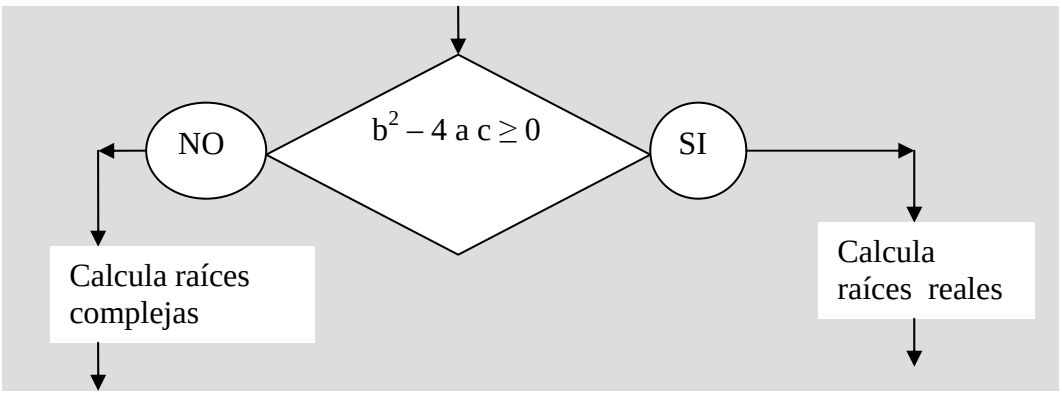
↑	Ctrl+p	Recupera la línea previa
↓	Ctrl+n	Recupera la línea siguiente
←	Ctrl+b	Mueve atrás un carácter
→	Ctrl+f	Mueve un carácter hacia delante
Ctrl+→	Ctrl+r	Mueve una palabra hacia la derecha
Ctrl+←	Ctrl+l	Mueve una palabra hacia la izquierda
Home	Ctrl+a	Va hasta el comienzo de la línea
End	Ctrl+e	Va hasta el final de la línea
Esc	Ctrl+u	Borra la línea
Del	Ctrl+d	Borra el carácter en la posición del cursor
Backspace	Ctrl+h	Borra el carácter anterior al cursor
	Ctrl+k	Borra al final de la línea

EXPRESIONES LOGICAS

Operadores relacionales

Hasta ahora, hemos tratado con operadores aritméticos matriciales o elemento a elemento, pero no hemos tomado decisiones dependiendo de alguna comparación.

Considérese el cálculo del discriminante del polinomio de segundo grado a x^2 + bx + c = 0, para averiguar si se calcularán raíces complejas o reales



la expresión $b^2 – 4\ a\ c \geq 0$ tiene sólo dos resultados posibles: verdadero o falso, y constituye, por lo tanto, una expresión lógica. El símbolo ≥ (mayor o igual, o "greater equal than") es un **operador relacional.** Las variables lógicas se pueden crear asigándoles el valor de una expresión lógica como por ejemplo: V_L = $b^2 – 4\ a\ c \geq 0$, que valdrá 1 si es verdadera la expresión y cero si es falsa.

El lenguaje de programación de MATLAB dispone de los siguientes operadores relacionales:

< menor que
> mayor que
<= menor o igual que
>= mayor o igual que
== igual que
~= distinto que*
*

El símbolo ~ se obtiene escribiendo char(126) en el command window

Si bien la utilización efectiva de estos operadores, como de los operadores lógicos, se realizará en el capítulo siguiente, resulta importante indicar que en Matlab los operadores relacionales pueden aplicarse a valores individuales (escalares de 1 x 1) o, quizás con menos frecuencia en ingeniería química, a vectores y matrices. Eso hace que tengan un significado especial.
Al igual que en C o Fortran, si una comparación se cumple el resultado es 1 (***true o verdadero***), mientras que si no se cumple es 0 (***false o falso***). Recíprocamente, cualquier valor distinto de cero es considerado como ***true*** y el cero como false. La diferencia con los otros lenguajes está en que cuando los operadores relacionales de MATLAB se aplican a dos matrices o vectores del mismo tamaño, *la comparación se realiza elemento a elemento*, y el resultado es otra matriz de unos y ceros del mismo tamaño, que recoge el resultado de cada comparación entre elementos.

Considérese el siguiente ejemplo como ilustración de lo que se acaba de decir:

```
>> A=[1 2;0 3]; B=[4 2;1 5];
>> A==B

ans =
0 1
0 0

>> A~=B

ans =
1 0
1 1
```

Al comparar matrices con operadores relacionales en Matlab, se obtienen matrices de resultados lógicos, que tiene elementos verdaderos (unos) y falsos (ceros). Resultados totalmente verdaderos implicaran unos en todos los elementos, mientras que los totalmente falsos incluirán ceros en todos los elementos

LOS OPERADORES RELACIONALES COMPARAN VALORES NUMERICOS PARA DAR RESULTADOS LOGICOS

Operadores lógicos

Supóngase que al resolver la raíz de una ecuación no lineal, (fx= $x^3 + 2x^2 + 2x + 2 = 0$, cuya única raíz real es -1,544) debe programar un algoritmo de "prueba y error" que, para continuar calculando, deba satisfacer que el valor absoluto de la función (fx) sea mayor o igual que una tolerancia(por ej, 0.01), y que el número de iteraciones, nit (es decir, de pruebas realizadas), sea menor o igual que un máximo predeterminado, nitmax (porque algunos métodos en ocasiones no convergen). Tal expresión debería vincular dos expresiones lógicas que incluyen operadores relacionales

1) **abs (fx) ≥ tol**
2) **nit ≤ nitmax**

Como los resultados de ambas comparaciones son valores lógicos, necesitamos un operador que compare valores lógicos, es decir un "Operador lógico".

Los operadores lógicos de MATLAB son los siguientes:

& and
| or
~ not (negación lógica)

Los operadores lógicos se combinan con los relacionales para comprobar el cumplimiento de condiciones múltiples
Por ejemplo, la condición múltiple 1)-2) será entonces, **abs (fx) ≥ tol & nit ≤ nitmax**
Entonces, los operadores lógicos vinculan expresiones lógicas para entregar resultados lógicos. Si el valor absoluto de fx es mayor o igual que la tolerancia (verdad), y el número de iteraciones es menor o igual que el máximo (verdad), entonces, el resultado global vinculado por el & resulta verdadero.

Se vio un ejemplo escalar, veamos ahora ejemplos de tipo matricial
Se verá un ejemplo

```
>> A= zeros (2) + 1.5
A =
   1.5000    1.5000
   1.5000    1.5000
>> B= ones(2)
B =
    1     1
    1     1
```

```
 A>1.2 & A>B

ans =

    1    1

    1    1
```

Es decir, los elementos de la matriz A son todos mayores que 1.2, y a su vez mayores que todos los de la matriz B, entonces, tenemos dos resultados lógicos verdaderos producidos por expresiones lógicas de operadores relacionales. El operador lógico & (and) combina dos resultados verdaderos, de manera que la expresión lógica global resulta verdadera también.

LOS OPERADORES LOGICOS VINCULAN EXPRESIONES LOGICAS PARA ENTREGAR RESULTADOS LOGICOS

"Tabla de verdad" Resultados de expresiones lógicas dependiendo del valor de los datos

Datos		Resultado de las expresiones lógicas		
a	b	~a **(not a)**	a&b (a and b)	a |b (a or b)
F	F	T	F	F
F	T	T	F	T
T	F	F	F	T
T	T	F	T	determinar

determinar si a or b es true cuando ambos son true. Implica determinar si el or es inclusivo o exclusivo. Por ejemplo considérese los siguientes datos y expresión lógica

```
>> a=1; b=2;  a >0 |a < b
ans =
    1
```

Prioridad de los operadores Se pueden construir expresiones con cualquier combinación de operadores aritméticos, de relación y lógicos. Los niveles de prioridad determinan el orden de evaluación de una expresión. Dentro de un mismo nivel de prioridad, la evaluación se realiza de izquierda a derecha. Las reglas de prioridad para los operadores MATLAB se muestran en este recuadro.

1. Paréntesis ()
2. Transposición(.'), potenciación (.^), transposición compleja conjugada('), potenciación de matrices(^)
3. signo más (+), signo menos (-), **Operador Lógico de negación NOT (~)**
4. Multiplicación (.*), división derecha (./), división izquierda(.\), multiplicación matricial (*), división matricial (/), división matricial izquierda (\)
5. suma (+), resta (-)
6. operador dos puntos (:)
7. Menor que (<), menor o igual que (<=), mayor que (>), mayor o igual que (>=), igual que (==), distinto (~=)
8. Operador lógico de conjunción AND (&) y el Operador lógico de decisión alternativa OR (|) tienen igual prioridad estableciéndose ésta de izquierda a derecha.

FUNCIONES DE BIBLIOTECA (LIBRERÍAS)

MATLAB tiene un gran número de funciones incorporadas. Algunas son ***funciones propias*** ("built-in functions"),esto es, funciones incorporadas en el propio código ejecutable del programa. Estas funciones son particularmente rápidas y eficientes. Existen además funciones definidas en ficheros ****.m*** que vienen con el propio programa o que han sido aportadas por usuarios de Matlab. Estas funciones extienden en gran manera las posibilidades del entorno para el desarrollo de aplicaciones.

Las funciones tienen nombre, argumentos y valores de retorno. Obsérvese que

[maximo, imax] = **max**(x);

Esta función, aplicada a una matriz genérica da el valor del elemento máximo y su posición. En matrices da los máximos de las columnas, así como el índice de su posición en ellas.

En los ejemplos que siguen, hay tantos valores de retorno como elementos de arreglo tengan *x*, *y,* y alfa.

r = **sqrt**(x^2+y^2);
a = **cos**(alfa) - **sin**(alfa);

Estas funciones trabajan con arreglos, elemento a elemento. MATLAB tiene diversos tipos de funciones. A continuación se enumeran los tipos de funciones más importantes, clasificadas según su finalidad:

1.- Funciones matemáticas elementales.
2.- Funciones especiales.
3.- Funciones matriciales elementales.
4.- Funciones matriciales específicas.
5.- Funciones para la descomposición y/o factorización de matrices.
6.- Funciones para análisis estadístico de datos.
7.- Funciones para análisis de polinomios.
8.- Funciones para integración de ecuaciones diferenciales ordinarias.
9.- Resolución de ecuaciones no-lineales y optimización.
10.- Integración numérica.
11.- Funciones para procesamiento de señal.

Los argumentos verdaderos de estas funciones pueden ser expresiones y también llamadas a otra función. MATLAB provee un gran número de funciones matemáticas estándar elementales, incluyendo abs (valor absoluto), sqrt (raíz cuadrada), exp, y sin. Tomar la raíz cuadrada o el logaritmo de un número negativo no es un error como en los otros lenguajes; en MATLAB se obtendrá el resultado complejo en forma automática. MATLAB también ofrece muchas más funciones matemáticas avanzadas, incluyendo las de Bessel y gamma. La mayoría de estas funciones aceptan argumentos complejos. Para disponer de una lista de las funciones elementales, se tipea, en el Command Window. **>>help elfun**

Se dan unos pocos ejemplos de **funciones elementales** que operan sobre en modo de arreglos (elemento a elemento), lo que incluye valores individuales, como sin(x), cos (x), etc.

Trigonométricas

sin(x), seno; cos(x), coseno; tan(x), tangente; asin(x), arco seno; acos(x), arco coseno; asinh(x), arco seno hiperbólico; acosh(x), arco coseno hiperbólico; atanh(x), arco tangente hiperbólica.

Exponenciales
log(x), logaritmo natural; log10(x), logaritmo decimal; exp(x), función exponencial; sqrt(x), raíz cuadrada

Funciones de operación algebraica
sign(x) devuelve -1 si x<0; 0 si x=0 y 1 si x>0. Aplicada a un número complejo, devuelve un vector unitario en la misma dirección.

rem(x,y) (de "remainder") provee el resto de la división (2 argumentos, dividendo y divisor); round(x) redondeo hacia el entero más próximo; fix(x) trunca los decimales dejando la cifra entera; gcd(x) máximo común divisor; lcm(x) mínimo común múltiplo

Funciones que actúan sobre vectores

Las siguientes funciones actúan sobre vectores (no sobre matrices ni sobre escalares)
[xm,im]=max(x) máximo elemento de un vector. Devuelve el valor máximo **xm** y la posición que ocupa **im;** **min(x)** mínimo elemento de un vector. Devuelve el valor mínimo y la posición que ocupa; **sum(x)** suma de los elementos de un vector; **mean(x)** valor medio de los elementos de un vector; **std(x)** desviación típica; **prod(x)** producto de los elementos de un vector; **n = length(x),** provee el número de elementos del vector x.
[y,i]=sort(x) ordenación de menor a mayor de los elementos de un vector **x**. Devuelve el vector ordenado **y**, y un vector **i** con las posiciones iniciales en **x** de los elementos en el vector ordenado **y**.

En realidad estas funciones se pueden aplicar también a matrices, pero en ese caso se aplican por separado a cada columna de la matriz, dando como valor de retorno un vector resultado de aplicar la función a cada columna de la matriz considerada como vector. Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz traspuesta.

Funciones que actúan sobre matrices

Se dieron en "el capítulo de manejo matricial". No obstante, se muestra aquí una función sumamente útil para realizar estructuras repetitivas
[m,n] = size(A) devuelve el número de filas **m** y de columnas **n** de una matriz
n = length(A) provee el número de elementos de la mayor dimensión de la matriz, sea ésta fila o columna (por ejemplo en una matriz de 2 x 5 devuelve 5, mientras que en una de 3 x 2 devuelve 3)

Para tener una lista completa de las funciones más avanzadas y de las funciones matriciales, se tipea **>> help specfun o >> help elmat**

Existen funciones especiales que proveen constantes útiles.

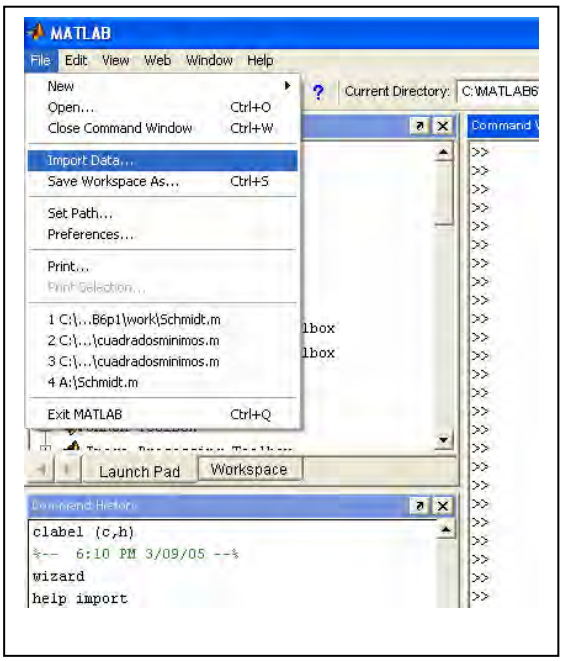
pi	3.14159265...
i	Unidad imaginaria , $\sqrt{-1}$
j	Igual que i
eps	Precisión relativa en punto flotante, 2^{-52}
realmin	Valor mínimo de punto flotante, 2^{-1022}
realmax	Valor máximo de punto flotante, $(2-\varepsilon)2^{1023}$
Inf	Infinito
NaN	No es un número (Not-a-number)

Infinito es generado dividiendo un número distinto de cero por un número igual a cero, or al evaluar expresiones matemáticas definidas que den *overflow*, esto es, que excedan realmax. NaN se genera al tratar de evaluar expresiones como 0/0 or Inf-Inf que no tienen valores matemáticos definidos.

Entrada y salida de datos (básico)

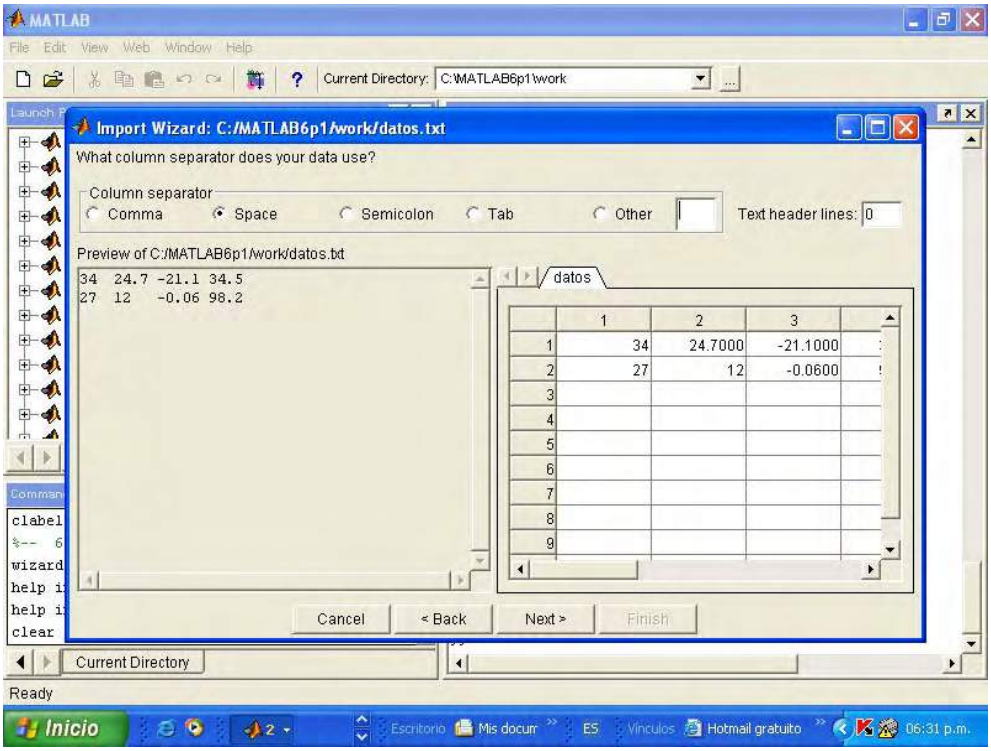
Import Wizard

Una forma sencilla de leer datos desde MATLAB en muchos formatos, incluyendo planillas de Excel, es utilizar el *Import Wizard (Asistente de la importación de datos)*



Archivo Ascii

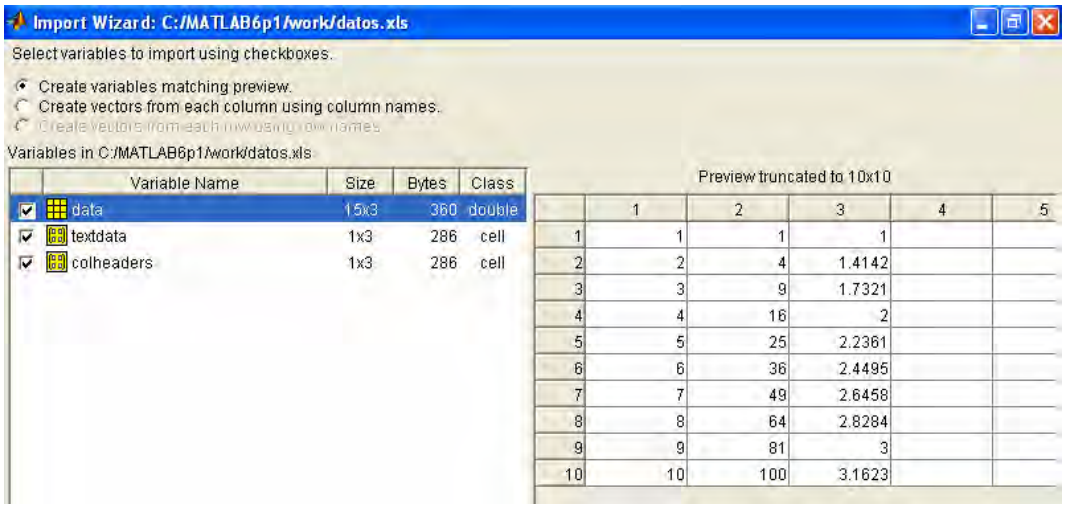
En *File, import data*, se elige el archivo, por ejemplo un archivo ASCII *datos.txt*, y luego el asistente va guiando al usuario. Al oprimirse "import data",se descuelga un cuadro de diálogo que muestra una "vista previa" del archivo a importar, y de cómo Matlab está interpretando los información a leer. Por ejemplo, si el archivo es numérico, se muestran éstos a efectos de que el usuario verifique si la separación entre los números es correcta, o no, o bien, si asigna todos los números a una sóla matriz o si el usuario prefiere asignar las variables por columna, a vectores. Veamos primero un ejemplo, de un archivo de datos hecho con un bloc de notas, que contiene dos líneas de 4 números cada una.



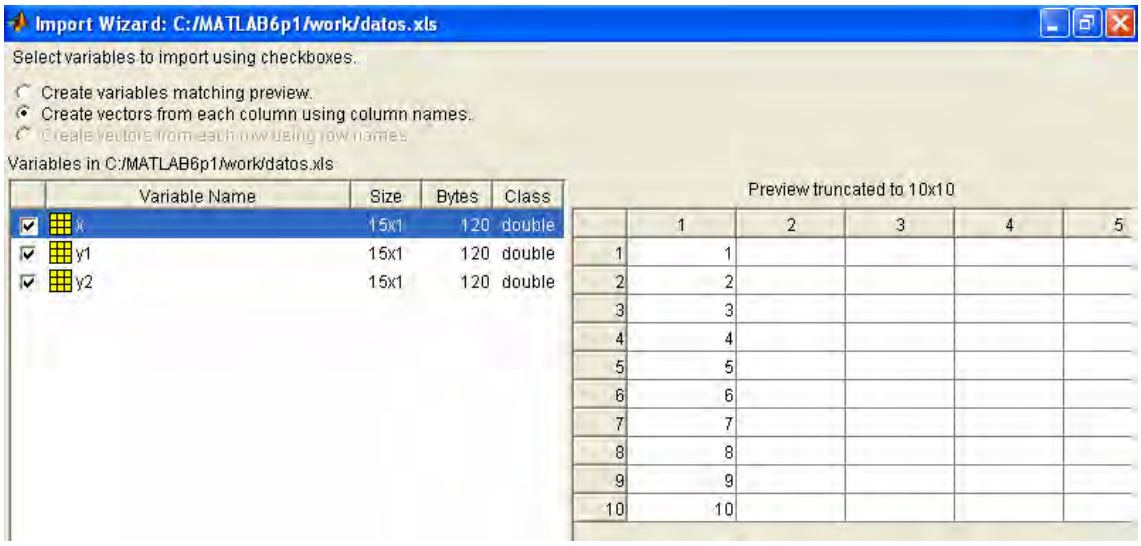
Los números en el archivo habían sido separados por espacios. El *Import Wizard* lo interpreta (ver opción tildada), pero da la opción para que el usuario lo defina también.

Planilla de cálculo

En el caso de una simple planilla de cálculo *Excel* con números, supongamos que tenemos una columna con valores de x, otra con sus cuadrados, llamada y1, y una tercera con las raíces cuadradas, llamado y2. Si la primera fila contiene los títulos de las columnas, es decir "x", "y1", e "y2", respectivamente, veamos que hace el *Import Wizard* en este caso.



el asistente detecta una matriz de números, a la que llama por defecto "data" de 15 filas por 3 columnas (obsérvese que muestra sólo 10 filas), y dos arreglos de celdas, que en realidad son coincidentes, uno de ellos "textdata" de 1 fila por tres columnas, conteniendo los nombres "x", "y1" e "y2", como todo texto que encuentra en el archivo. El otro arreglo de celdas, "colheaders", corresponde al encabezamiento de las columnas. Es decir, si hubiera habido más texto en el archivo, el arreglo "textdata" resultaría más grande que el "colheaders". Obsérvese también que permite elegir entre "aceptar" la organización de variables propuesta por Matlab, o bien, crear variables con el contenido de cada columna. Si elegimos esta última opción

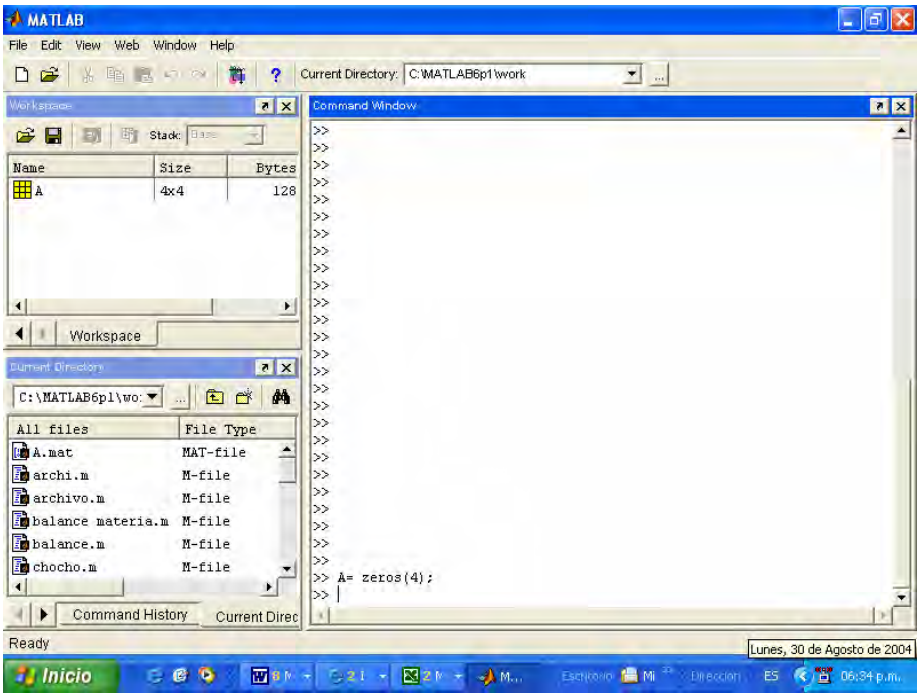


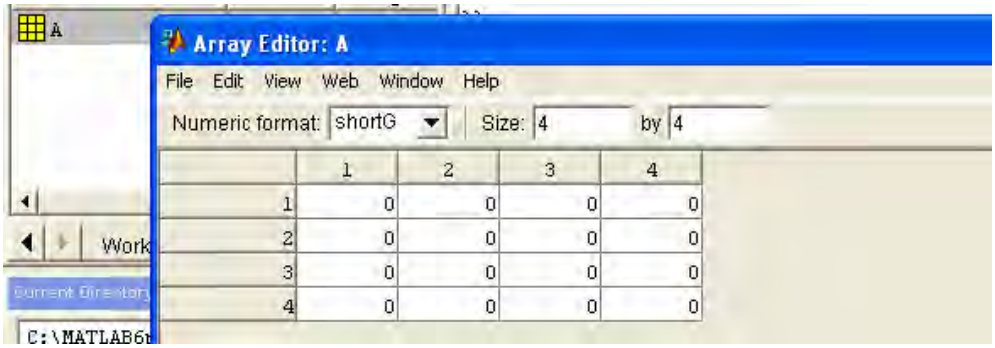
la propuesta conlleva a tres vectores columna, de 15 filas. Como está marcado x, ése es el vector que muestra en el preview, pero sólo 10 elementos.

La desventaja del Import Wizard es que no puede usarse directamente desde un programa Matlab tipo **nombredearchivo.m**, ni desde el command window sino solamente desde el menú. En consecuencia, su utilización implicará importar "manualmente" los datos, antes de la ejecución del programa

Ingreso de datos con el editor de arreglos de Matlab

Se generan "los moldes", matrices con el mismo tamaño que las que queremos ingresar, simplememente para luego tipear los datos. Por ejemplo, si debemos ingresar una matriz de 4 x 4, se puede prearmar la estructura, con una matriz de ceros
>> A=zeros(4);
>> Luego se va al editor de arreglos para abrir la matriz A creada



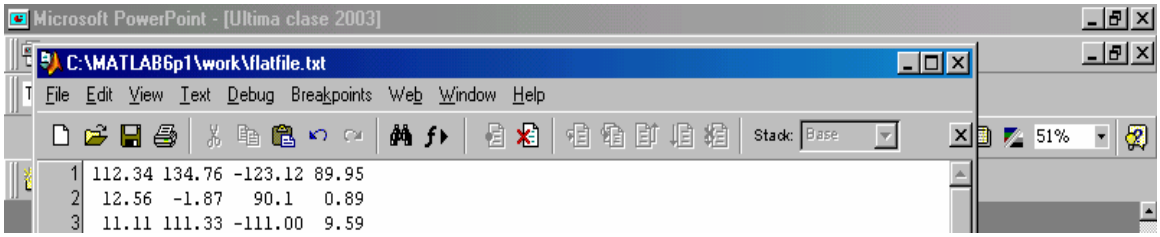


Con el editor de arreglos a la vista, se pueden retipear como se desee los elementos de la matriz con los datos verdaderos.

Otras posibilidades

-Se puede utilizar el Copy y Paste del portapapeles para traerlos a un archivo *.m. En esta forma los datos se pueden editar y se pueden asignar a variables.

-Se puede crear cargar un archivo ASCII con filas de longitud constante separadas por **Enter,** y varios datos por fila separados por blancos. Los mismos, en un archivo *.txt se leen desde Un programa de MATLAB (*.m) o desde la línea de comandos,empleando una comando **load.** Supongamos que el archivo ***flatfile.txt*** se va a leer de esta manera.




```
>>load ('flatfile.txt');

>> flatfile
flatfile =
  112.3400  134.7600 -123.1200   89.9500
   12.5600   -1.8700   90.1000    0.8900
   11.1100  111.3300 -111.0000    9.5900
```

Guardar sesión y copiar salidas: Comando *diary*

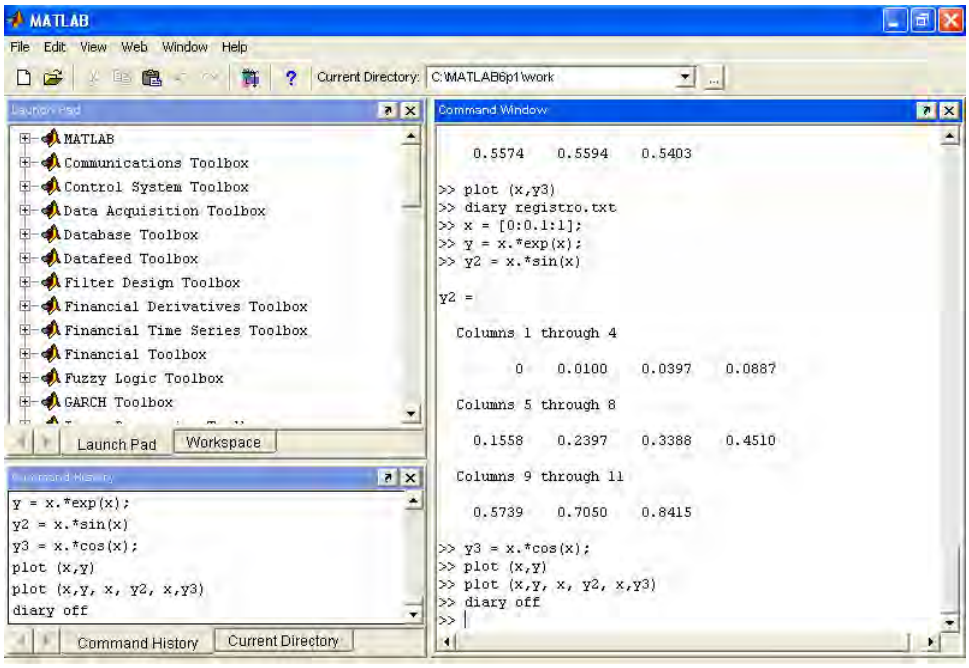
Los comandos ***save*** y ***load*** crean y cargan archivos binarios o ASCII con el estado de la sesión. Existe otra forma más sencilla de almacenar en un archivo un texto que describa lo que el programa va haciendo (la entrada y salida de los comandos utilizados). Esto se hace con el comando ***diary*** ("agenda") en la forma siguiente:

>> diary nombredearchivo.txt

...

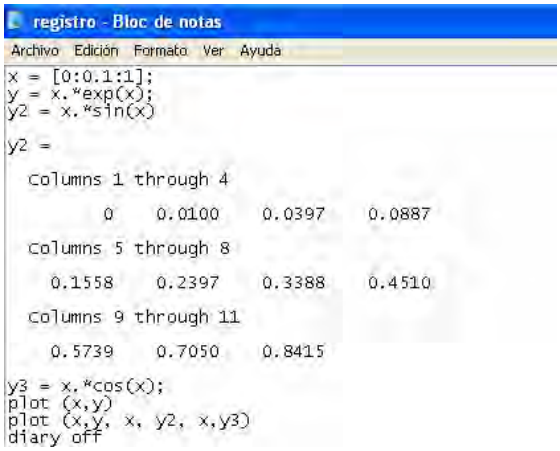
>> diary off

El comando ***diary off*** suspende la ejecución y ***diary on*** la reanuda. El simple comando ***diary*** pasa de ***on*** a ***off*** y viceversa. Para poder acceder al archivo ***filename.txt*** con ***Bloc de Notas*** o ***Word*** es necesario que ***diary*** esté en ***off***. Si en el comando ***diary*** no se incluye el nombre del archivo se utiliza por defecto un archivo llamado ***diary*** (sin extensión).

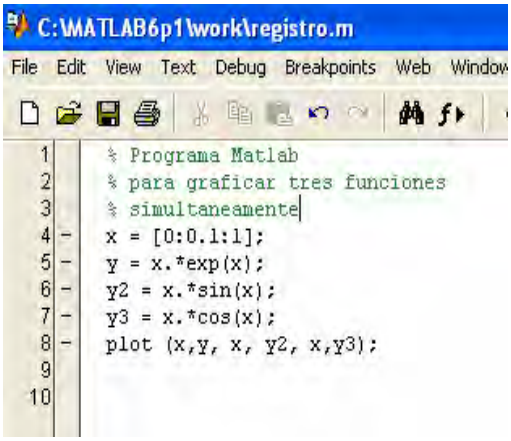


Como generar un archivo *.m a partir de otro archivo *.txt generado con el comando "diary".

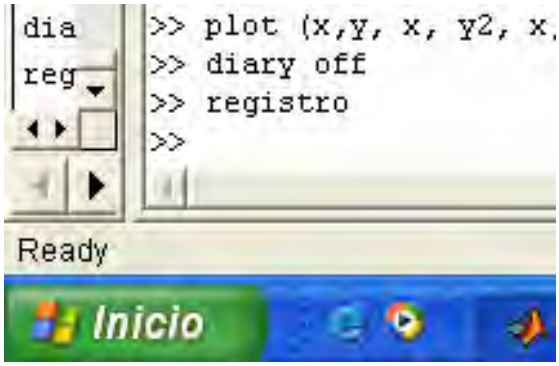
Para generar un pequeño programa Matlab archivo *.m, luego de usar los comandos "diary", podemos leer el archivo "registro.txt" en el Bloc de Notas. Vemos que, para producir un gráfico con las tres funciones juntas, deberíamos borrar la visualización automática de y2, poner ";" luego de esta variable justamente para que no se visualice, borrar la sentencia plot(x,y), y la instrucción "diary off". Así, seleccionamos todos los contenidos de "registro.txt", abrimos un archivo "registro.m", copiamos y luego borramos lo mencionado previamente.

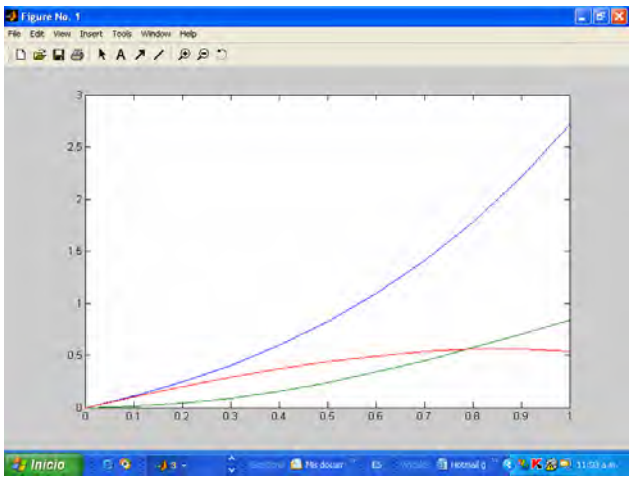


Así, saliendo del bloc de notas, yendo al Matlab (hay que mantener todos los programas abiertos), haciendo "File, New, M-file", copiando la información y procediendo de la manera planeada, se obtiene el archivo "registro.m" siguiente:



La ejecución en el Command Window permite obtener





Ejecutar un archivo *.m que incluye el nombre de otro archivo *.m que contiene los datos

Para crear matrices, se puede usar también archivos *. m . Por ejemplo, se puede crear un archivo que contenga estas líneas.

dosarreglos.m

```
A = [ ...
   1.0    3.0   -2.0     13.0;
   5.0  -10.0    1.0      8.0;
   2.0    6.0   -7.0      1.0;
   4.0   -1.5    1.0      2.0 ];

   b = [9;
        4;
       -3;
        2];
```

Guarde este archivo con el nombre **dosarreglos.m** . Se puede crear otro programita *.m que, si A es la matriz de los coeficientes de un sistema de ecuaciones lineales y B el vector de los términos independientes, calcule las incógnitas del sistema. Lo llamaremos **incognitas.m**

incognitas.m

```
% programa que calcula incógnitas de un sistema de ecuaciones lineales

dosarreglos; % lee las matrices A y B del archivo "dosmatrices.m"

ink= A\b % calcula las incognitas del sistema lineal
```

Al ejecutar el nuevo programa

```
>> incognitas
>>ink=
   0.0388
   0.2946
   0.7984
   0.7442
```

que nos da las cuatro incógnitas del problema.

Ingreso y Salida de datos interactivo para programas Matlab *.m

Este tipo de instrucciones son muy utilizadas para crear programas donde el usuario interactúa con el programa al tiempo de ejecución.

Sentencias Input y Disp.

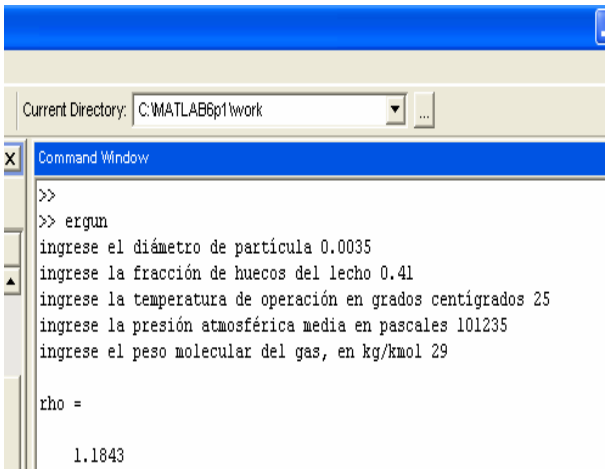
```
>> No = input('Ingrese el número de Avogadro ')
Ingrese el número de Avogadro 6.02e23
No =
  6.0200e+023
>> disp ('Numero de Avogrado = '), No
Numero de Avogrado =
No =
  6.0200e+023
```

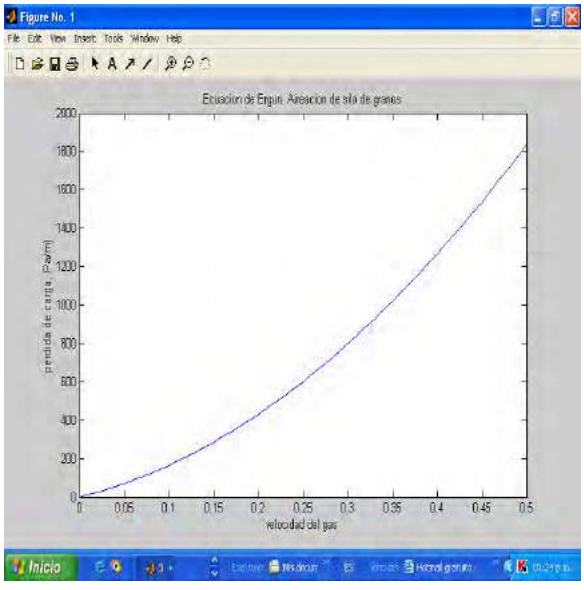
Integración de conceptos

Se realizarán dos archivos *.m, uno para convertir temperaturas en grados Farenheit a centígrados "conversion.m" y el otro "ergun.m" para calcular la pérdida de carga de un gas a través de un lecho relleno, un rango de velocidades.

```
% Programa para convertir °F a °C
TF =input('ingrese la temperatura en grados Farenheit');
TC = (TF-32)*5/9;
Disp ('para una temperatura en grados Farenheit de'), TF
Disp ('la temperatura en grados centígrados es'), TC
```

```
% Programa para la ecuación de Ergun
dp =input('ingrese el diámetro de partícula  ');
epsi =input('ingrese la fracción de huecos del lecho  ');
Tc = input ('ingrese la temperatura de operación en grados centígrados  ');
p= input ('ingrese la presión atmosférica media en pascales  ');
M= input('ingrese el peso molecular del gas, en kg/kmol  ');
Tk = Tc + 273.16; Rg = 8314;
% cálculo de la densidad del gas
rho = p*M/(Rg*Tk)
% Calcule la viscosidad del gas (aire)
vis = 1.735e-5 + 4.138e-8*Tc;
% defino el rango de velocidades en el vector v
v = 0:0.01:0.5;
coef_L = 150.*vis.*(1-epsi).^2/(dp.^2*epsi.^3);
coef_T = 1.75.*rho.*(1-epsi)/(dp*epsi.^3);
%calcula la pérdida de carga por unidad de longitud de lecho relleno
Deltap_L= coef_L .*v + coef_T.*v.^2;
plot (v, Deltap_L)
title ('Ecuacion de Ergun. Aireacion de lecho relleno de partículas ')
xlabel('velocidad del gas, m/s')
ylabel('perdida de carga, Pa/m')
```





Otro ejemplo de integración de conceptos: Cálculo de la temperatura de ebullición en La Paz, Bolivia, a 3650 m de altitud s.n.m.

Ecuaciones

$p = p_0 - \rho g h$ donde ρ es la densidad del aire, p la presión barométrica a altitud h, p0 la presión barométrica s.n.m, M el peso molecular del aire (29 kg/kg mol), R la constante de los gases (8314 J/kg mol K) $\rho = \frac{pM}{RT}$

Así, se llega a

$$p = \frac{p_0}{1 + \frac{M\ g\ h}{RT}}$$

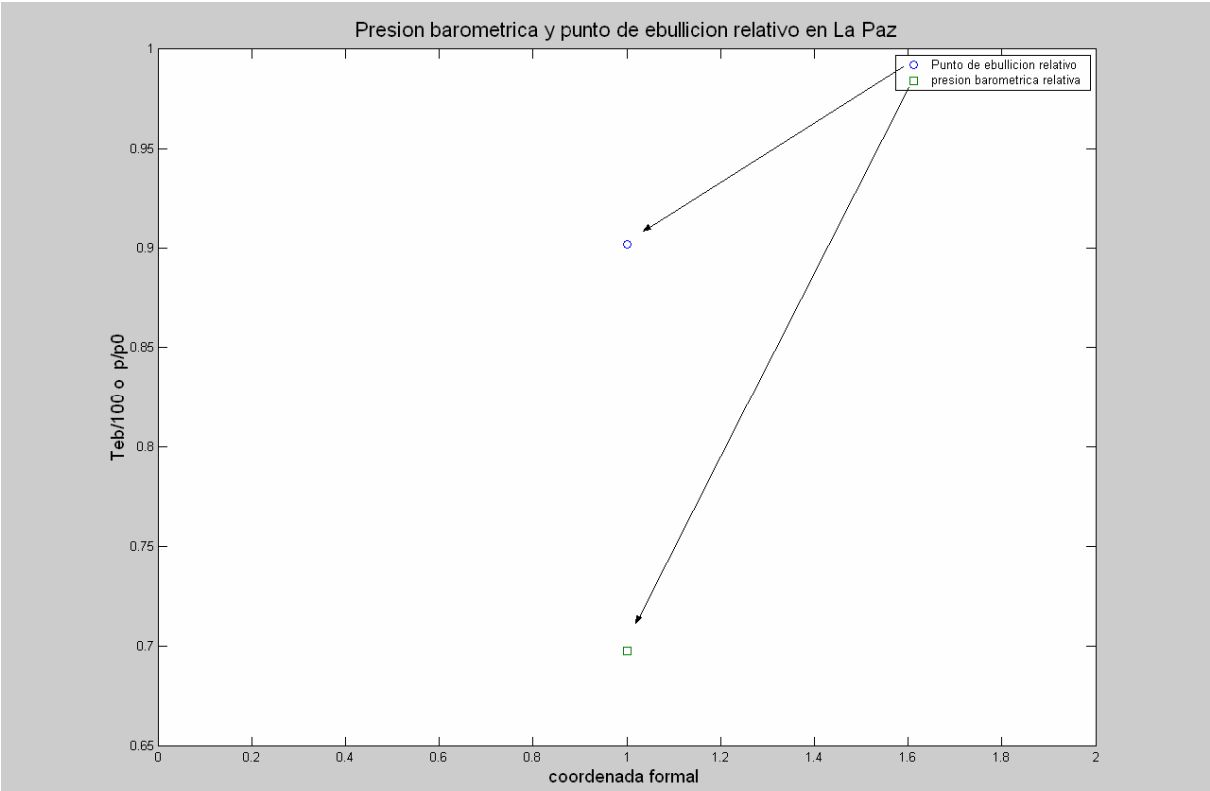
Por otra parte, la ecuacion de presion de vapor de sustancias puras de Antoine, se expresa así

$$\log_{10} p_s = A - \frac{B}{T + C}$$

donde p_s es la presión de vapor del agua en mmHg, y T la temperatura en ºC. Los coeficientes para el agua, se obtuvieron en el rango de 60 a 150 ºC (Felder, R M y Rousseau, R W "Principios Elementales de los Procesos Químicos, 3º Edición, Limusa Wiley, 2004, página 644). El programa realizado y los resultados son los siguientes

```
% Programa para calcular el punto de ebullicion del agua en La Paz, Bolivia
% a 3650 m de altitud s.n.m
clear
% constantes de la ec. de presion de vapor del agua de Antoine,
%Log10 ps = A - B/(T + C) % donde ps esta en mmHg, y T en ºC

A = 7.96681; B= 1668.210; C = 228.00; Ma = 29;
g = 9.81; R = 8314; Tamb_m = 15;
h = 3650; p0 = 101325; % Pascales

% Calculo de la presi´on atmosferica en la paz, utiliizando densidad variable
% y una temperatura media de 15ºC
p = p0/(1+Ma*g*h/(R*(Tamb_m+273.16)));
disp ('presion barometrica en La Paz, en atmosferas'), p./p0
x = 1
pmmHg = p.*760./101325;
Teb = B/(A- log10 (pmmHg)) -C;
y1 = p./p0; y2 = Teb./100;
plot (x, y2,'o', x, y1,'s' )
xlabel ('coordenada formal ', 'Fontsize', 14)
ylabel ('Teb/100 o  p/p0',  'Fontsize',14 )
Title ('Presion barometrica y punto de ebullicion relativo en La Paz', 'Fontsize', 16)
legend ('Punto de ebullicion relativo', 'presion barometrica relativa' )
```

Los resulados fueron p/p_0 = 0.6976, y Teb = 90.2ºC, de manera que Teb/100 es 0.902. Es decir, la presión barométrica se reduce un 30%, mientras que el punto de ebullición, sólo un



10%. A que se debe la diferencia entre ambos valores relativos?

Ejercicio de autoevaluación

Resuelva la ecuación cuadrática en Matlab, $ax^2 + bx + c = 0$, ingresando la terna de datos de coeficientes mediante la sentencia interactiva "input". Aclare los resultados obtenidos con la sentencia "disp". Utilice archivos *.m.

ANEXO

Funciones preprogramadas para cálculos con polinomios

Para MATLAB un polinomio se puede definir mediante un vector de coeficientes. Por ejemplo, el polinomio:

$$x^4 - 8x^2 + 6x - 10 = 0$$

se puede representar mediante el vector [1, 0, -8, 6, -10]. MATLAB puede realizar diversas operaciones sobre él, como por ejemplo evaluarlo para un determinado valor de **x** (función ***polyval()***) y calcular las raíces (función ***roots()***):

```
>> pol=[1 0 -8 6 -10]
pol =
1 0 -8 6 -10
>> roots(pol)
ans =
-3.2800
2.6748
0.3026 + 1.0238i
0.3026 - 1.0238i
>> polyval(pol,1)
ans =
-11
```

Para calcular producto de polinomios, se utiliza la función ***conv()*** (de *producto de convolución*). Se va a ver ahora cómo se multiplica un polinomio de segundo grado por otro de tercero:

```
>> pol1=[1 -2 4]
pol1 =
1 -2 4
>> pol2=[1 0 3 -4]
pol2 =
1 0 3 -4
>> pol3=conv(pol1,pol2)
pol3 =
1 -2 7 -10 20 -16
```

Para dividir polinomios existe otra función llamada ***deconv()***. Las funciones orientadas al cálculo con polinomios son las siguientes:

-poly(A) polinomio característico de la matriz **A**
-roots(pol) raíces del polinomio **pol**
-polyval(pol,x) evaluación del polinomio **pol** para el valor de **x**. Si **x** es un vector, **pol** se evalúa para cada elemento de **x**
-polyvalm(pol,A) evaluación del polinomio **pol** de la matriz **A**
-conv(p1,p2) producto de convolución de dos polinomios **p1** y **p2**
[-c,r]=deconv(p,q) división del polinomio **p** por el polinomio **q**. En **c** se devuelve el cociente y en **r** el resto de la división
-residue(p1,p2) descompone el cociente entre **p1** y **p2** en suma de fracciones simples (ver ***>>help residue***)
-polyder(pol) calcula la derivada de un polinomio
-polyder(p1,p2) calcula la derivada de producto de polinomios
-polyfit(x,y,n) calcula los coeficientes de un polinomio **p(x)** de grado **n** que se ajusta a los datos **p(x(i)) ≅ y(i)**, utilizando el criterio de cuadrados mínimos (no el de interpolación).
-interp1(xp,yp,x) calcula el valor interpolado para la abscisa **x** a partir de un conjunto de puntos dado por los vectores **xp** e **yp**.
interp1(xp,yp,x,'m') como el anterior, pero permitiendo especificar también el método de interpolación.

CAPITULO 6. PROGRAMACION MATLAB II. Control del flujo de información.

PROGRAMAS DE ESTRUCTURA SECUENCIAL EN MATLAB

Estructura secuencial pura

Hacia el final de la clase Programación I, hemos visto dos archivos *.m, realizados por el usuario en el editor de programas de Matlab, uno de ellos convertía temperaturas de una escala a otra, muy simple y con estructura secuencial. Esta estructura es la más sencilla de programación y **consiste en ejecutar las instrucciones en el mismo orden en que están escritas, pasando el flujo de información una sola vez por cada instrucción.** Aquí va otro ejemplo de estructura secuencial pura, que calcula el sueldo de un empleado, en función de las horas trabajadas y la paga horaria.

```
% Programa que calcula el sueldo mensual de un empleado
Nombre = input(' Ingrese Nombre del empleado ');
Horas = input('Ingrese horas trabajadas por semana ');
Pagahor= input('Ingrese paga horaria ');
%{Se calculará el sueldo mensual}
Sueldo = Horas * Pagahor*4.5;
%{ Se visualiza el sueldo}
disp (' se visualizara el nombre del empleado y su sueldo'), Nombre, Sueldo
```

Una particularidad de Matlab: las estructuras pseudosecuenciales

También vimos un programa para calcular la ecuación de Ergun de los lechos rellenos, que predice la pérdida de carga de un fluído por unidad de longitud de un dado lecho relleno en función de la velocidad, y de las características del lecho y del fluído, como porosidad, diámetro de partícula, densidad y viscosidad. Ese programa parecía estar hecho en estructura secuencial, dado que no tenía decisiones que tomar, pero, era realmente un programa puramente secuencial, o Matlab, en su estilo de ocultar complejidad, disimulaba una estructura repetitiva?. Se verá una fracción del mismo nuevamente

```
v = 0:0.01:0.5;
coef_L = 150.*vis.*(1-epsi).^2/(dp.^2.*epsi.^3);
coef_T = 1.75.*rho.*(1-epsi)/(dp.*epsi.^3);
%calcula la pérdida de carga en el  lecho relleno
Deltap_L= coef_L .*v + coef_T.*v.^2;
plot (v, Deltap_L)
```

Las sentencias en negrita nos muestran en realidad estructuras repetitivas, de variables vectorizadas, v y Deltap_L. No obstante, la naturaleza matricial de Matlab disimula su complejidad a tal punto que la ecuación de Ergun parece calcular sólo escalares.

Se puede considerar entonces que Matlab dispone de un nuevo tipo de estructuras: las pseudosecuenciales, que son en realidad estructuras secuenciales puras mezcladas con estructuras repetitivas totalmente vectorizadas. No son estructuras secuenciales puras por que el control del flujo de información pasa tantas veces por las instrucciones en negrita como valores tengan los elementos del vector v.

Controlando el Flujo de Ejecución de las Instrucciones

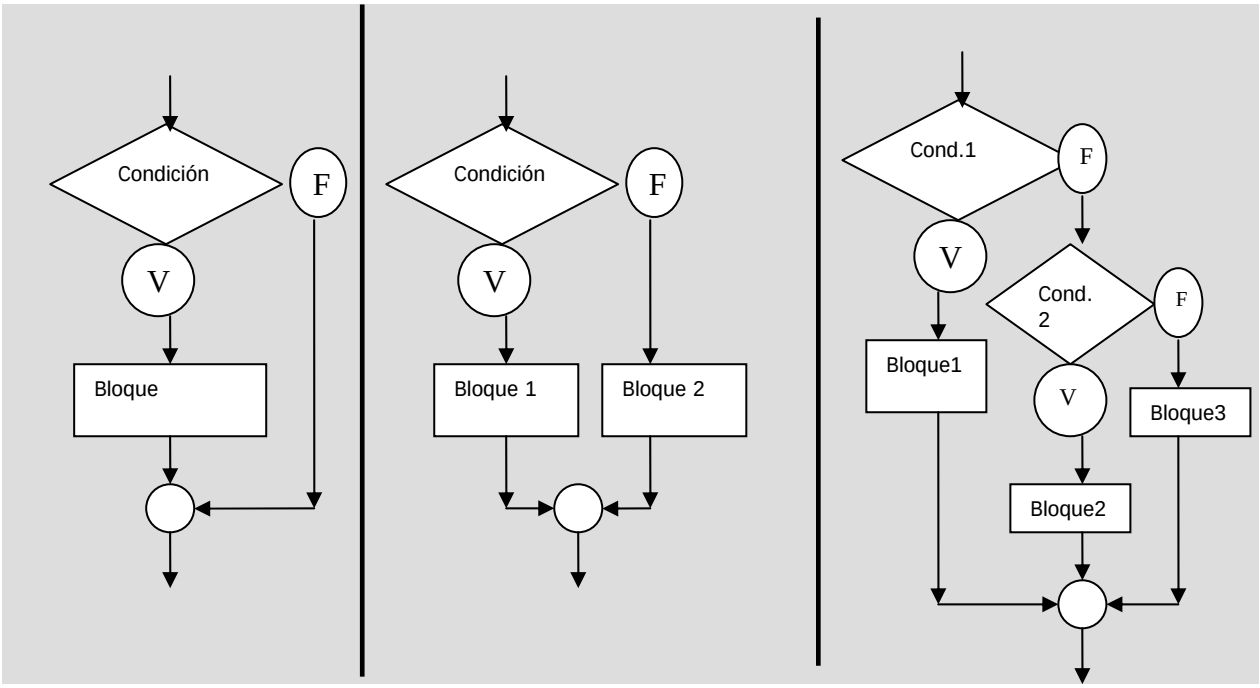
El tipo de programa más elemental es aquel con Estructura Secuencial como "sueldo.m". Estos programas, como tales, no son de gran utilidad, debido a que el flujo de la información pasa una vez sola por cada instrucción, y no se toman decisiones en el programa.
Como se ha visto en la parte de Diagramación, se pueden programar estructuras para decidir entre entre una, dos o más alternativas, y estructuras repetitivas, para métodos de prueba y error (búsqueda de raíces) u operaciones de arreglos. Las estructuras repetitivas pueden tener control numérico o lógico.

MATLAB tiene varias estructuras (*constructs*) de control del flujo de información

if (estructura alternativa)
switch (estructura alternativa)
for (lazo o bucle repetitivo, con control numérico)
while (lazo o bucle repetitivo, con control lógico)
continue (instrucción)
break (instrucción)

IF
La instrucción **if** evalúa una expresión lógica y ejecuta un grupo de sentencias cuando la expresión es verdadera (*true*). Las palabras claves opcionales **elseif** y else permiten la ejecución de grupos alternativos de sentencias. La palabra clave **end,** que guarda correspondencia con el **if**, termina el último grupo de sentencias. Los grupos de instrucciones están delineados por las cuatro palabras claves indicadas. No aparecen corchetes ni llaves. En la figura se muestran los casos de selección simple (izquierda), doble (centro) y múltiple (derecha)

Selección Simple	Selección doble	Selección múltiple



La selección simple

Como ejemplo, veamos un segmento de programa para identificar una situación de flujo laminar

```
Re= input (' numero de Reynolds')
if  Re <= 2100
        fanning = 64./Re
        disp ('flujo laminar, factor = '), fanning
end
```

La estructura alternativa de selección simple implica que si la expresión lógica es falsa, el programa sigue su curso como si no existiera el if

La selección doble

Si se desea diferenciar los flujos laminares de los no laminares, entonces se puede usar el if-then-else

```
Re= input (' numero de Reynolds')
if  Re <= 2100
        fanning= 64./Re
        disp ('flujo laminar, factor = '), fanning
else
        fanning = 0.0791./Re.^0.25
        disp ('flujo no laminar, factor = '), fanning
 end
```

La selección múltiple

Cuando existen más de dos posibilidades, cobra mayor utilidad la regla de que el número de preguntas que se hacen es n-1, donde n es el número de posibilidades. Para esto se emplea la cadena if-elseif- else- end

```
Re= input (' numero de Reynolds')
if  Re <= 2100
        fanning= 64/Re
        disp ('flujo laminar, factor = '), fanning
elseif  Re <= 1e5
        fanning = 0.0791/Re.^0.25
        disp ('flujo de transición, factor =  '), fanning
 else
        fanning = 0.0044
        disp ('flujo turbulento, factor constante ='), fanning
 end
```

los tres casos son alternativos, uno sólo de ellos se cumple

EJERCICIO. REALICE EL PROGRAMA DEL CALCULO DEL SUELDO DEL EMPLEADO. Considere que si trabaja menos de 140 horas mensuales, el sueldo se factura igual que antes, multiplicando directamente la paga horaria por el número de horas trabajadas. En cambio, si supera las 140 horas semanales, la diferencia entre el total y 140 se factura con un 30% de aumento.

Ejemplo programado de estructura alternativa: Resolución de una ecuación cuadrática

```
clear, clc;
%    Programa para resolver una ecuacion cuadratica ax^2 + bx +c = 0
%    Mediante metodo analitico
%    Lee coeficientes de la ecuacion
a = input ('ingrese coeficiente a ');
b = input ('ingrese coeficiente b ');
c = input ('ingrese coeficiente c ');

disc = b.^2 -4.*a.*c;

if disc >=0
   x1r = (-b + sqrt (disc))./(2.*a);
   x2r = (-b - sqrt (disc))./(2.*a);
else
   x1r = -b./(2.*a);
   x1i = sqrt (-disc)/(2.*a);
   x2r = x1r;
   x2i = -x1i;
end
disp ('las raices son x1r, x1i, x2r, x2i '), x1r, x1i, x2r, x2i;
```

Otros ejemplos con el uso del bloque de If

Ejemplo: determinación de la divisibilidad por dos y por tres
clear

```
clear
numero = input('Ingrese un número entero: ' );
resto2 = rem(numero,2);
resto3 =  rem(numero,3);

if resto2==0 & resto3==0
        disp ('su número es divisible por 2 y 3')
else
        if resto2==0
                disp (‘su numero es divisible por 2, pero no por 3’)
        else
                if   resto3==0
                        disp ('su number is divisible por 3 pero no por 2')
                else
                        disp ('su numero no es divisible ni por 2 ni por 3')
                end
        end
end
```

Más casos de selección múltiple

Ejemplo de registro de conductor

En algunos países, se extiende un permiso para manejar autos a partir de los 16 años, y a partir de 18, el "registro de conductor", que deberá renovar cada 5 años hasta los 70 de edad. A partir de los 70 años, necesitará una licencia especial, teniendo que aprobar nuevamente un examen de manejo, y revisiones de vista y audición. Se le pide a un programador que haga un programa interactivo, a efectos de que los ciudadanos de un determinado país, y los extranjeros legalizados puedan conocer como es el sistema de registros de conductor. El mismo, luego de haber sido organizado en un diagrama de bloque puede ser como sigue:

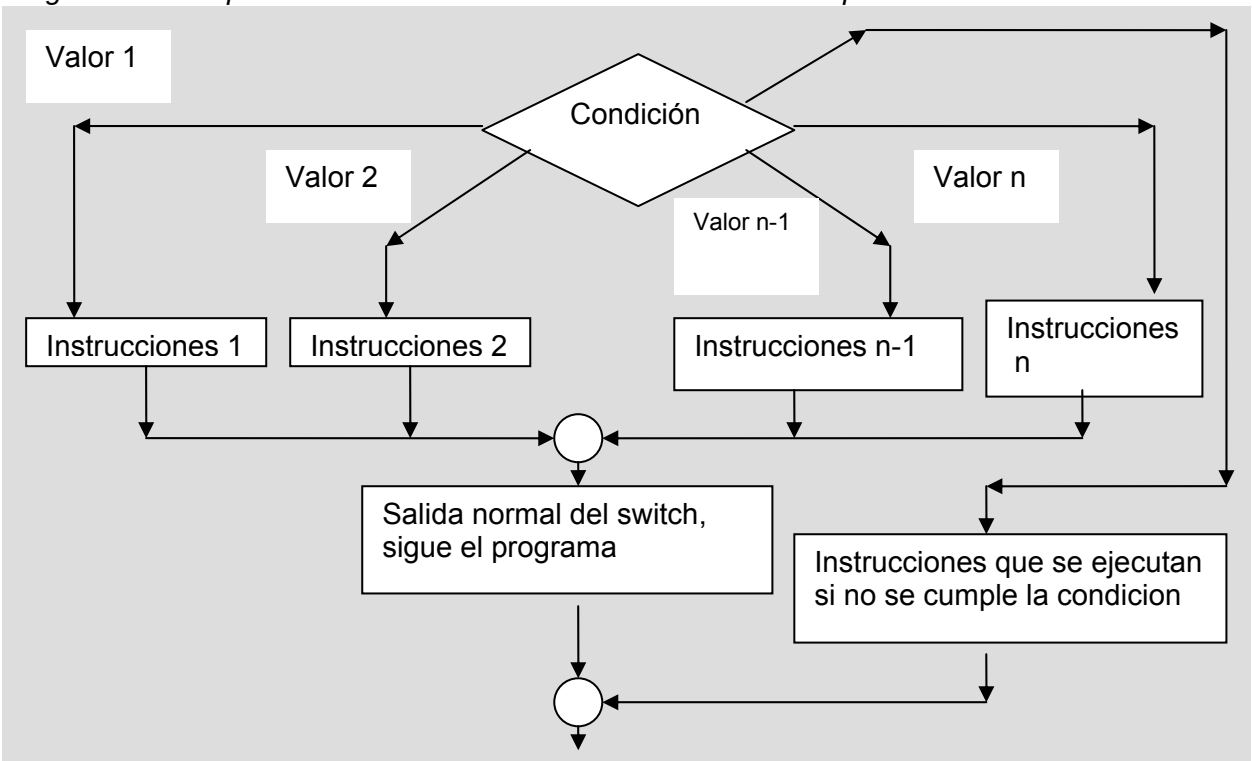
```
edad = input ('Ingrese edad del postulante ')
if edad < 16
        disp (' Deberá esperar a cumplir los 16 años para pedir el permiso')
else if  < 18
        disp (' Puede solicitar un permiso inicial para manejar')
else if < 70
        disp(' Puede solicitar el registro de conductor clásico')
else
        disp (' Deberá solicitar un permiso especial')
end
```

Este programa funciona bien, pero muchas situaciones de selección múltiple pueden programarse con la estructura switch.

switch

La proposición switch ejecuta grupos de instrucciones basados en el valor de una variable o expresión, cuyo resultado debe ser un escalar o una cadena de caracteres. Las palabras clave case y otherwise delimitan los grupos. Se ejecuta solamente el primer caso de coincidencia. Siempre debe haber un end para corresponder un switch.

Diagrama de bloques de una sentencia switch en selección múltiple



Ejemplo: Supóngase que un empleado, de acuerdo a su situación familiar, recibe un ingreso diferente, siendo éste igual al sueldo de la categoría del escalafón si es soltero sin hijos (0 personas a cargo), de un 30% mayor si es casado o vive en pareja (1 persona a cargo), 40% mayor si tiene un hijo menor de 21 años (2 personas a cargo), 50% mayor si tiene dos hijos menores (3 personas a cargo), 60% mayor si tiene tres hijos menores (4 personas a cargo) y que debe plantear su caso si tiene más de tres hijos. Realice un programa que calcule el sueldo a recibir, recibiendo como dato el sueldo del escalafón

```
sueldo_esc = input (‘ingrese sueldo del escalafón para su categoría ‘)
Numero_a_cargo = input (‘ ingrese el número de personas a cargo  ‘)
switch (Numero_a_cargo)
        case 0
                Sueldo_corregido = sueldo_esc;
        case 1
                Sueldo_corregido = 1.3.* sueldo_esc;
        case 2
                Sueldo_corregido = 1.4.*sueldo_esc;
        case 3
                Sueldo_corregido = 1.5.*sueldo_esc
        case 4
                Sueldo_corregido = 1.6*sueldo_esc
 otherwise

        If Numero_a_cargo > 4
                disp('Inicie un expediente planteando su caso’)
        else
                disp (‘dato equivocado, ingrese otra vez el número’)
        end
end
```

ESTRUCTURAS REPETITIVAS

Como se ha visto en diagramación, las estructuras repetitivas permiten ejecutar varias veces un determinado bloque de instrucciones, para distintos valores de determinadas variables. Realmente le dan sentido a la programación, y permiten escribir programas concisos que realizan muchos cálculos. En Matlab, se tiene la estructura ya vista “tipo rango” para asignar valores a vectores, que es muy concisa y oculta complejidad, por ejemplo:

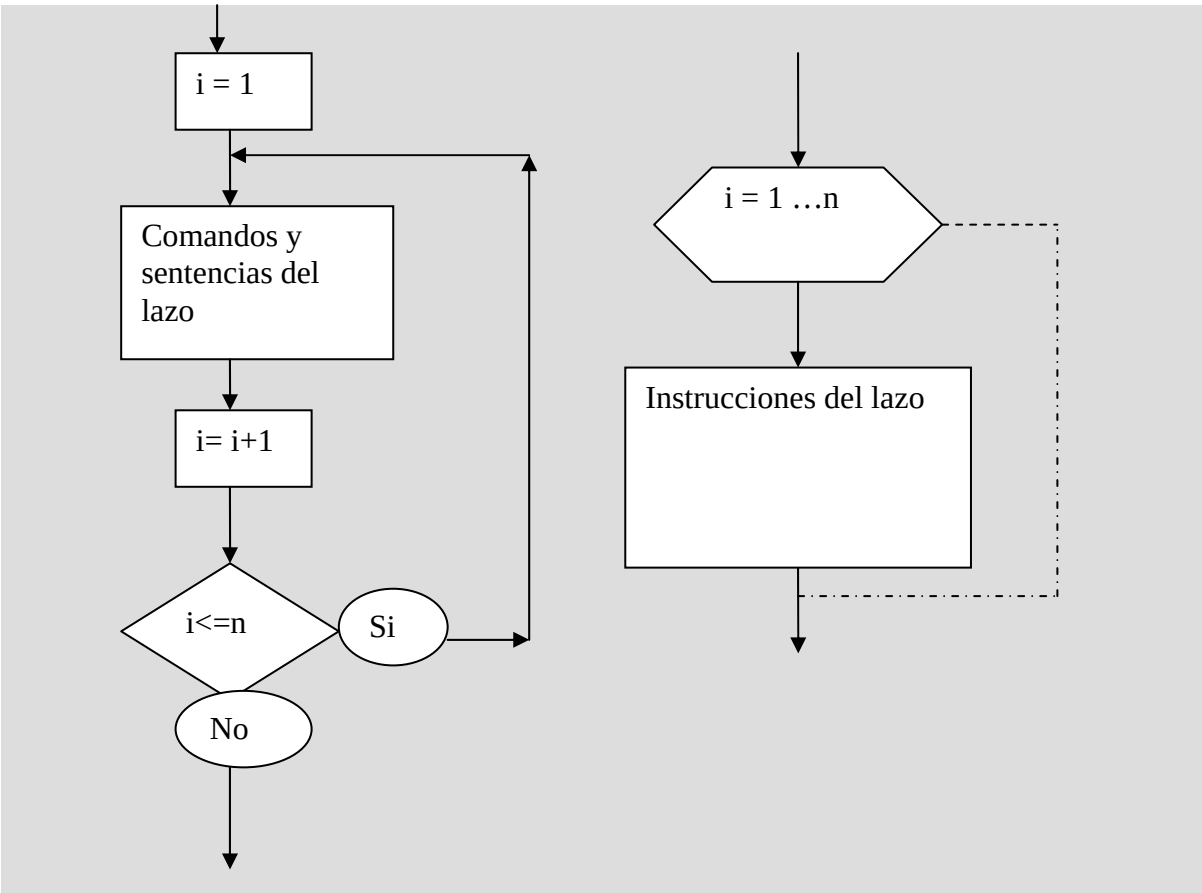
x = 0: 0.01: 10; % esta sentencia asigna 1001 elementos al vector x.

Matlab ejecuta estas **instrucciones vectorizadas** con gran eficiencia, y deben usárselas toda vez que se pueda.

No obstante, en los casos donde haya que procesar algunos elementos del arreglo o matriz, y no todos, o bien en casos donde la vectorización pueda resultar dificultosa, se utilizan los **lazos for**.

Estructura repetitiva con control numérico for

Formas alternativas de representarla en diagramas de bloques



for

El lazo for repite un grupo de instrucciones un número fijo, predeterminado de veces. La estructura se delimita con el end correspondiente.

En este caso la proposición for repite las sentencias para valores de i desde 1 hasta n, variando de a uno.

```
for i=1:n
  sentencias
end
```

Es una buena idea "dejar sangría" en los lazos para mejorar la legibilidad, especialmente cuando están anidados.

En este caso que sigue, el índice va tomando los valores de un vector

```
valoresdevector = [1, 3, 4, 5, -4, 10]
for i=valoresdevector
  sentencias
end
```

La sintaxis general es

```
for i=valor_inicial : incremento : valor_final
   sentencias
end
```

El ejemplo de vectorización dado antes, hecho con un for, es algo más difícil debido a que hay que saber cuantos valores va a tener el vector que todavía no se armó. Como va de 0 a 10 de a 0.01, se supone que tendría 101 hasta 1, y finalmente, 1001 valores hasta 10.

```
for i = 1:1001
        x(i)= (i-1)*0.01
end
```

Obsérvese que siempre es preferible trabajar con valores enteros, los subindices del arreglo, y no con los valors de los elementos del arreglo como índices del for

Por ejemplo, para el ejemplo anterior, si usáramos x como índice del for

```
for x =0:0.01:10
        disp (genero un vector x usando sus valores como índice del for ?)
end
```

Esa pregunta, formulada 1001 veces en el programa, nos lleva a una respuesta: **no**, el valor de x que se obtiene al final es 10, un escalar.

Si se persiste en usar los valores de los elementos y no los índices, se puede nombrar otra variable ("valor", por ejemplo) con esos datos, y hacer

```
i = 1
for valor = 0:0.01:10;
        x(i) = valor;
        i = i+1
end
```

con lo que usaremos for y no necesitaremos de antemano saber cuantos elementos tendrá el vector a generar.

Ejemplo con valores decrecientes

Ejemplo,

```
for i = n: -0.2:1
        sentencias
end
```

el lazo se ejecuta la primera vez con i = n, y luego i se va reduciendo de 0.2 en 0.2 hasta que llega a ser menor que 1, donde se termina el ciclo.

Otro ejemplo

Genere un vector a partir de otro, cuyos índices serán calculados en base a los del vector previo

```
for i = 1:n
        j = i.^2 – 10

        if j >0
                Y(j) = x(i)
        else
                j
                disp (' para ese valor de j no hay asignación posible')
        end
end
```

Lazos anidados

Por ejemplo, si se asume que, por ejemplo, m = 5 y n = 4, crear la matriz de Hilbert, usando zeros para preasignar la matriz:

```
% matriz de Hilbert
A = zeros(m,n)  % Preasignar la matriz acelera los cálculos
for i = 1:m
   for j = 1:n
      A(i,j) = 1/(i+j-1);
   end
end
A
```

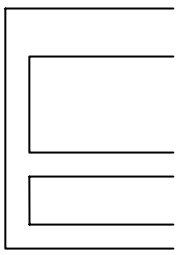
Sin preasignación de "A" con la matriz zeros, Matlab hubiera debido "estirar" la matriz al calcular cada elemento nuevo, lo cual le hubiese demandado tiempo.
los resultados son.

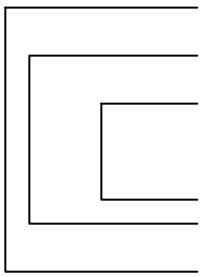
```
A =
   1.0000    0.5000    0.3333    0.2500
   0.5000    0.3333    0.2500    0.2000
   0.3333    0.2500    0.2000    0.1667
   0.2500    0.2000    0.1667    0.1429
   0.2000    0.1667    0.1429    0.1250
```

No se podría crear la matriz de Hilbert con cálculos vectorizados, dado que hay que manejar los subíndices uno por uno, y no el arreglo entero.

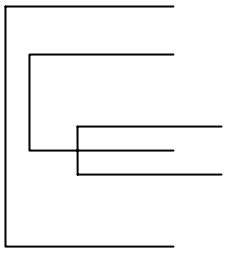
Los lazos anidados de **for** no pueden cruzarse, siempre debe estar el mas interno totalmente contenido en el mas externo. El anidamiento puede incluir incluso mas de dos lazos

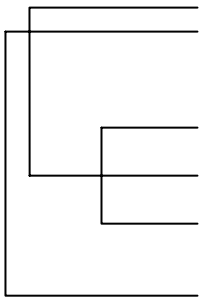
EJEMPLOS PERMITIDOS





EJEMPLOS NO PERMITIDOS





Cuando la velocidad sea importante, sin embargo, siempre se deben contemplar las posibilidades de vectorizar sus algoritmos.

Ejemplo de integración de conceptos. Resolución analítica de ecuaciones de segundo grado, para varias ternas de valores a, b, c. Estructura repetitiva alternativa y secuencial.

```
%  Programa para resolver una ecuacion cuadratica ax^2 + bx +c = 0.
%  Mediante metodo analitico
%  Lee coeficientes de la ecuacion

% Se leen las ternas de una planilla de calculo

coeficientes = xlsread ('coeficientescuadraticos');

% se asignan a vectores, la primer columna es “a”, la segunda
% es “b” y la tercera es “c”

a = coeficientes (:, 1);
b = coeficientes (:, 2);
c = coeficientes (:, 3);

% calcula el numero de ternas

nter = length(a);

%Lazo repetitivo exterior, para controlar las ternas a utilizar

for k = 1:nter;

% Se calcula el discriminante

   disc(k) = b(k).^2 -4.*a(k).*c(k);

% se fuerza al programa a distinguir entre raíces reales y complejas
% conjugadas mediante una estructura alternativa de decisión doble

        if disc (k) >=0
                x1r(k) = (-b(k) + sqrt (disc (k))./(2.*a (k)); x1i(k) = 0;
                x2r(k) = (-b(k) - sqrt  (disc (k)))/(2.*a (k));  x2i(k) = 0
        else

                x1r(k)= -b (k)./(2.*a(k));
                x1i (k)= sqrt (-disc (k))./(2.*a (k));
                x2r (k)= x1r (k);
                x2i (k)=-x1i (k);

        end % bloque if-then-else

end % lazo for

% el programa almacena todos los resultados en los vectores x1r, x1i, x2r y x2i
save salidamuchascuadraticas x1r x1i x2r x2i -ascii
```

LAZO repetitivo con control lógico "while"

En la programación moderna, el lazo repetitivo FOR, con control numérico, resulta muy limitado. Puede usarse en matrices, trabajando elemento a elemento cuando es imprescindible, pero hay que conocer el tamaño del arreglo. En ingeniería química, es muy usual resolver problemas por métodos iterativos (secuencias de prueba y error para determinar un valor que no puede calcularse analíticamente) que converjan a una tolerancia (por ej. El método del punto medio) donde no puede estimarse a priori el número de repeticiones, como lo requiere el lazo FOR. Así, resulta más flexible introducir un tipo de lazo que se ejecute, mientras se cumpla una *condición* (generalmente una expresión lógica)

```
while condición
        sentencias
end
```

El lazo while repite un grupo de sentencias un número indefinido de veces bajo el control de una condición. La construcción termina con un end. Las sentencias se siguen ejecutando mientras la condición sea verdadera, es decir, mientras el resultado de la evaluación lógica de la condición sea distinto que cero.

Sea el ejemplo siguiente: un investigador debe deshidratar una muestra a una temperatura moderada, pero debe hacerlo en una estufa con un vacío de 700 mmHg. Cual es la mayor temperatura que podrá utilizar si se desea evitar la ebullición del agua durante el proceso?

Con este fin, se debe calcular el punto de ebullición del agua a la presión reinante en la estufa de vacío.

Procedimiento: si "el vacío" es de 700 mmHg, la presión reinante en la estufa será p= 760-700 = 60 mmHg.
La ecuación de presión de vapor del agua, que se dispone es la siguiente:

$$Ln\ p_s = A - \frac{B}{(T + 273.16)} - C\ Ln(T + 273.16)$$

donde ps es la presión de saturación del agua en Pa, y T es la temperatura en ºC. Para determinar el punto de ebullición, se debe considerar p_s = p = 60 x 101325/760 = 8000 Pa. Los coeficientes de la ecuación son: A = 54.119; B = 6547.1; C = 4.23. Si se despeja T, se puede obtener la ecuación de recurrencia para el método de prueba y error $T_c = f(T_s)$, es decir, se supone un valor de T, se calcula la expresión que sigue a la derecha del signo igual.

$$T_c = \frac{B}{\left(A - C.*Ln\left(T_s + 273.16\right) - Ln\ p\right)} - 273.16$$

y se asigna el resultado a la variable que está a la izquierda del signo igual. La coincidencia de un valor supuesto con el calculado, dentro de una tolerancia, hará que el cálculo se termine. En consecuencia, se puede establecer una CONDICION, que consista en verificar si la diferencia en valor absoluto entre T_c y T_s es mayor que 0,1ºC.

El programa escrito fue:

```
clear, clc
A = 54.119; B = 6547.1; C = 4.23; vac = 700;
pmmHg =  760 -vac;
p = pmmHg.*101325/760;
difer = 1; % este valor se coloca a propósito para "entrar" al ciclo while
Ts = 30; % valor inicial supuesto
niter = 1; % inicialización del contador de iteraciones
while difer >= 0.1
   Tc = B./(A - C.*log(Ts +273.16) - log (p))-273.16;
   niter = niter +1;
   difer = abs (Tc-Ts);
   Ts = Tc;
end
niter
Tc
```

```
>> niter =
    5
Tc =
   41.5527
```

El punto de ebullición del agua a una presión absoluta de 60 mm Hg, es de 41,6ºC. En consecuencia, al investigador le conviene deshidratar a una temperatura menor que esa para evitar la ebullición del agua en la muestra. Obsérvese que el ciclo while puede ejecutar un número de repeticiones desconocido de antemano, siendo así muy adecuado para el cálculo numérico iterativo.

Sentencias **break** y **continue**

La instrucción **break** termina la ejecución de un lazo while o for. En lazos anidados, termina la ejecución del lazo interior solamente. Si se usa **break** fuera de un lazo for o while in un programa matlab, se termina la ejecución del programa en este punto. También se termina el programa si el **break** se ejecuta dentro de una estructura if o switch-case

Por su parte, la instrucción **continue** pasa el control a la siguiente iteración de un lazo for o while en la cual aparece, permitiendo saltar las instrucciones que faltan terminar en ese lazo for o while. En lazos anidados, **continue** pasa el control a la siguiente repetición de los lazos for o while dentro de los cuales está escrito.

```
clear
n = 0;
while n < 10
  n = n+1
  a = input ('Ingrese un valor mayor que 0:  ');
  if (a<0)
    disp ('debe ingresar un número positivo ')
    disp (' este programa terminará ')
    break
  end
  disp ('el log natural de este número es ')
  disp (log(a))
end
```

```
clear
n = 0;
while n < 10
  n = n+1
  a = input ('Ingrese un valor mayor que 0:  ');
  if (a<0)
    disp ('debe ingresar un número positivo ')
    disp (' este programa terminará ')
    continue
  end
  disp ('el log natural de este número es ')
  disp (log(a))
end
```

Ensaye en Matlab estos programas, y encuentre sus resultados, a efectos de comprobar las descripciones precedentes de las instrucciones **break** y **continue**

CAPITULO 7.CONCEPTOS DE PROGRAMACIÓN ESTRUCTURADA Y MODULAR

Modularizacion de Programas

Hasta ahora, trabajamos con programas tipo script en archivos *.m (programas desarrollados en lenguaje Matlab) que alojan las variables en el mismo espacio de trabajo que el utilizado por el command window. Sin embargo, cuando los programas son largos, se dificulta su revisión y la detección de errores, incluyendo, los errores de tipeo que no violan las reglas del lenguaje Matlab, como por ejemplo

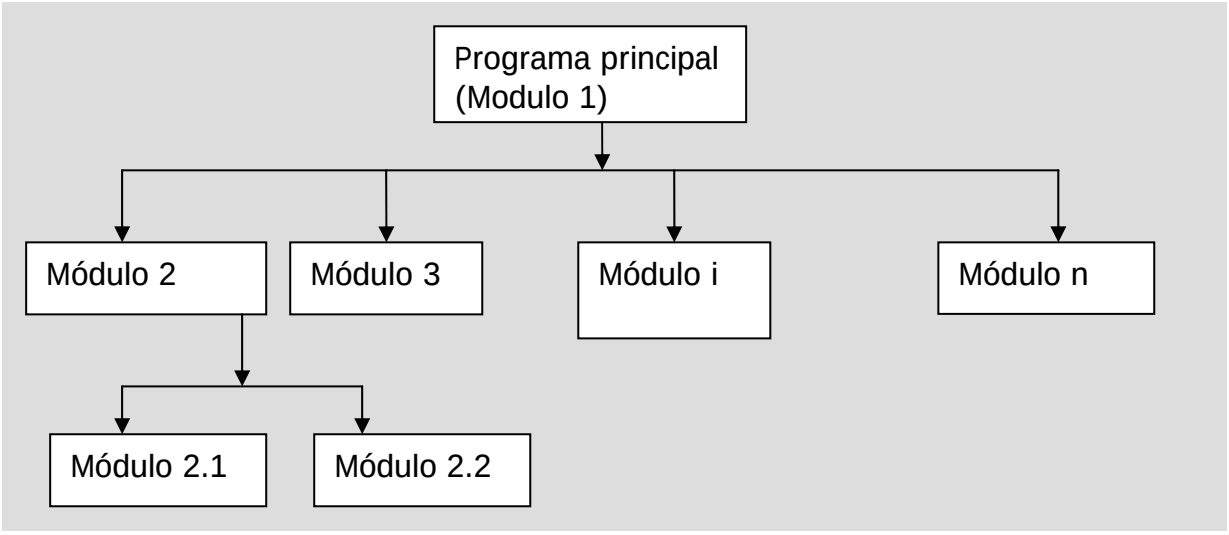
Fuerza = masa .*acel.^2

en lugar de

Fuerza = masa.*acel

Para reducir el número de errores, conviene considerar que un programa "largo" (una página o más) es una tarea compleja que puede dividirse en un número de tareas más simples vinculadas entre sí.

Con este fin, el programa total debe partirse en tareas bien definidas, específicas. Cada parte puede llamarse "módulo de programa". Generalmente el programa total consta de un programa principal que coordina las tareas y llama a los otros módulos de programa. Incluso puede haber módulos que llamen a su vez a otros módulos. Las modularización de las tareas suele seguir un orden jerárquico que puede planificarse desde lo más general a lo más específico (diseño descendente, o *Top-down*), o bien desde lo más específico a lo más general (diseño ascendente). Por ejemplo, una estructura diseñada en forma descendente, puede tener el aspecto siguiente:



Se puede considerar que excepto el programa principal, los restantes módulos de programa son **subprogramas**. El seguimiento de este método, se denomina "programación modular" y es la tendencia que ha seguido la programación desde la introducción de estos conceptos en la década de 1980.

Ventajas de la modularización de programas

-Permiten descomponer problemas complejos en varias tareas más simples

-Los módulos pueden reutilizarse en distintos programas

Programación estructurada en cada módulo de programa

Es lo que hemos estado viendo desde que comenzó el curso, pero ahora se van a enfatizar estos conceptos. La programación estructurada es una serie de técnicas que

-Promueve una programación clara, ordenada y documentada (con comentarios) , que facilita el trabajo en equipo, lo que es clave para los trabajos de ingeniería.

-Utiliza las estructuras secuenciales, alternativas y repetitivas, tratando de evitar, en lo posible, saltos incondicionales en el control del flujo de la información, como los que realizan las instrucciones "break" y "continue".

Recursos abstractos de la programación estructurada de cada módulo de programa

-En programación estructurada, la primera tarea que se realiza debe aparecer primero, y la última, en último lugar, de manera que se pueda seguir el flujo de información a lo largo del programa sin esfuerzos innecesarios.

-Se recomienda utilizar nombres de variables relacionados con su significado, que no sean innecesariamente largos

- Se recomienda la utilización de comentarios para aclarar el significado de algunas instrucciones

-Se recomienda utilizar mayúsculas para las matrices y minúsculas para los vectores a efectos de reconocerlos inmediatamente.

-Se sugiere dejar espacios entre términos de una ecuación, e incluso entre variables y operadores de multiplicación, división y potenciación, a efectos de mejorar la legibilidad del programa.

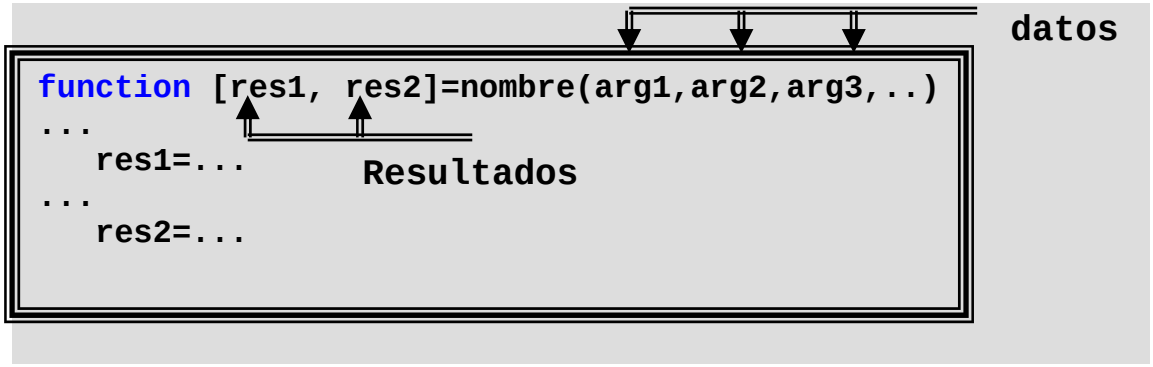
-Las instrucciones largas, por ejemplo aquellas que resultan de codificar ecuaciones complicadas, pueden dividirse en varios grupos de términos y/o factores, definidos previamente a la expresión final. Obsérvese que este recurso abstracto es, conceptualmente, el mismo que conduce a dividir un programa total largo en un programa principal y varios subprogramas relacionados

Se considera que la observación de estas reglas de estilo, lejos de inhibir la creatividad del programador, la favorecen pues vuelven la programación más eficaz. Los subprogramas de matlab se construyen mediante la instrucción "function"

Subprogramas realizados por el programador Matlab

Matlab permite escribir subprogramas o módulos de programa como funciones, realizadas por el programador, denominadas "functions".

Definicion general de function



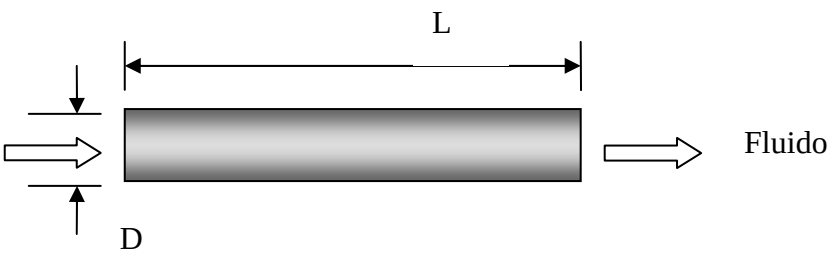
Tanto arg1 ... arg n, los datos recibidos por la function, como res1 ... resn, los resultados que calcula, pueden ser, cada uno de ellos, escalares (matrices de 1x1), o arreglos (vectores o matrices), de manera que la complejiad de las "functions" es altamente variable dependiendo del caso. Los resultados deben calcularse explícitamente antes de terminar el código de la function.

*Tipos de archivo *.m* Como se ha indicado, hay dos tipos de archivo *.m.

Archivo *. m de comandos (Script)	Archivo *.m tipo Function
No acepta argumentos de entrada ni retorna argumentos de salida	Puede aceptar argumentos de entrada y retornar argumentos de salida
Opera sobre datos del espacio de trabajo (workspace)	Por defecto, las variables internas son locales de la función
Utiles para automatizar una serie de pasos que necesita realizar muchas veces	Utiles para extender el lenguaje MATLAB a su aplicación específica

Como se llama a las functions?

Se verá un ejemplo donde se calcula la pérdida de presión Δp que experimenta un fluido al pasar por una cañería de longitud *L* y diámetro *D*, mediante la ecuaciónde Fanning



La ecuación de Fanning es:

$$\Delta p = f \, \tfrac{1}{2} \rho v^2 \, \frac{L}{D}$$

En un archivo llamado Deltap.m, se escribe el programa siguiente:

```
d = input ('ingrese el diámetro de la cañería ');
long = input ('ingrese la longitud de la cañería ');
rhof = input ('ingrese la densidad del fluido ');
vis = input ('ingrese la viscosidad del fluido ');
v = input ('ingrese la velocidad superficial del fluido ');

% se calculará el número de Reynolds

Re = rhof .*v.*d./vis

% se llamará a la función que calculará el factor de fricción de Fanning

f = Fanning (Re)

% se calculará la pérdida de presión por  fricción en cañerías rectas

Deltap = f .* 0.5 .* rhof .* v .^2 .*long ./ d ;

Disp (' La velocidad y pérdida de presión son  '), v, Deltap
```

y en otro archivo, Fanning.m (el nombre del archivo que contiene la "function" debe coincidir con el nombre de la misma)

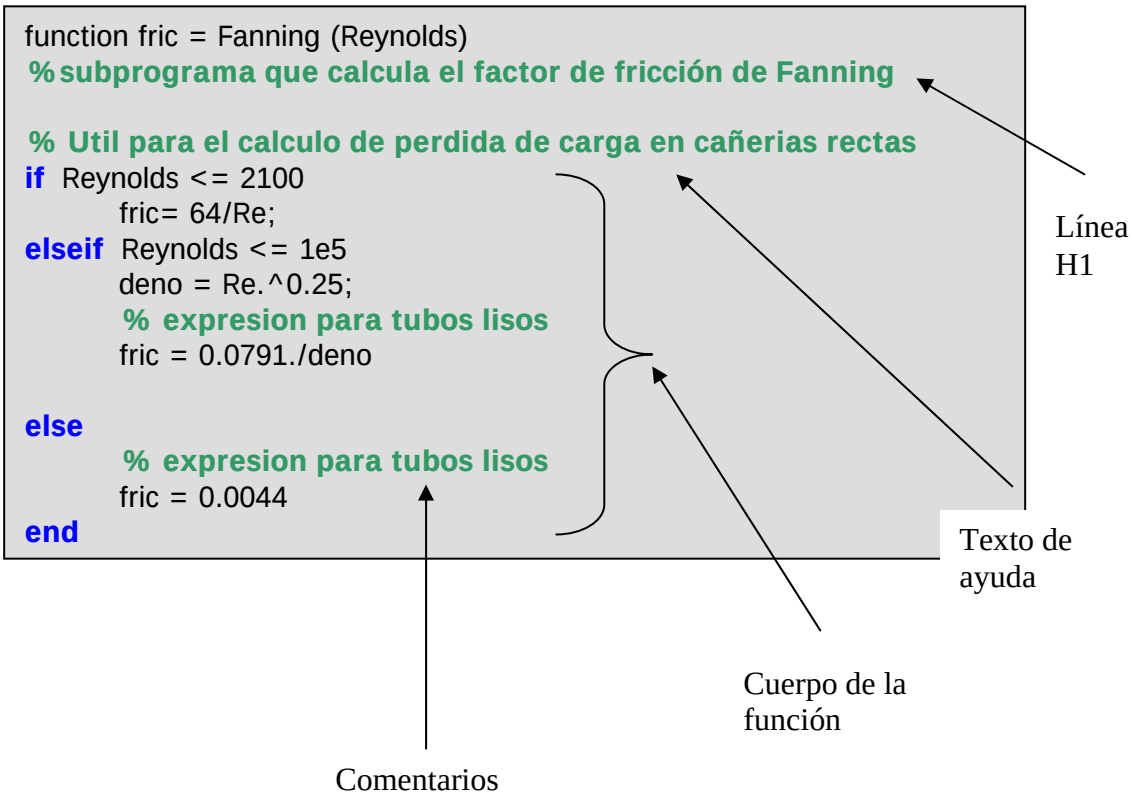
```
function fric = Fanning (Reynolds)
%subprograma que calcula el factor de fricción de Fanning

% Util para el calculo de perdida de carga en cañerias rectas
if  Reynolds <= 2100
        fric= 64/Re;
elseif  Reynolds <= 1e5
        deno = Re.^0.25;
        fric = 0.0791./deno
else
        fric = 0.0044
end
```

Obsérvese que el resultado de la función no fue encerrado entre corchetes por ser una sola variable (sea escalar o arreglo). Pruebe esta ecuación en Matlab para distintos fluidos. Si tiene dudas acerca de la velocidad a ingresar, imagine un caudal volumétrico y divídalo por la sección transversal de la cañería.

Partes básicas de un archivo M de Function

Un archivo M de function consiste de:

- La línea de definición
- La línea H1
- Texto de ayuda
- El cuerpo de la Function
- Comentarios



Variables locales y globales

Las variables de una "function" (de ahora en adelante, "función", excepto en la declaración de la misma) son siempre variables locales, y por tanto no pertenecen al espacio de trabajo del command window y los archivos *.m que no son funciones. En el caso del factor de fricción, se hizo lo que se recomienda: que los argumentos y resultados de la función no se llamen igual que los valores correspondientes en la llamada. Los valores deben corresponderse por ser datos del mismo tipo, por guardar el mismo orden en que se envían como argumentos, y se reciben como resultados, sólo que se llaman diferente en función y programa. Hay dos fuertes razones por las cuales conviene que los nombres no coincidan en llamada y función

1) Las variables de función son locales a la función y ocupan por tanto espacios de memoria diferentes que los del workspace, no siendo visibles desde este espacio de trabajo.
2) Las funciones pueden usarse junto a distintos programas principales, o pueden ser llamadas desde el command window en casos diferentes. Una nomenclatura igual a la de algún programa principal específico les puede quitar generalidad.

A efectos de dar un ejemplo para 1), vamos a ver que sucede cuando hay nombres de variables iguales en llamada y función

repitamos el segmento del programa principal donde se llama a la función

```
f = Fanning (Re)
```

y ahora escribamos la función con igual nomenclatura que en el programa

```
function f = Fanning (Re)
%subprograma que calcula el
%factor de fricción de Fanning
if  Re <= 2100
        f= 64/Re;
elseif  Re <= 1e5
        deno = Re.^0.25;
        f = 0.0791./deno
else
        f = 0.0044
end
```

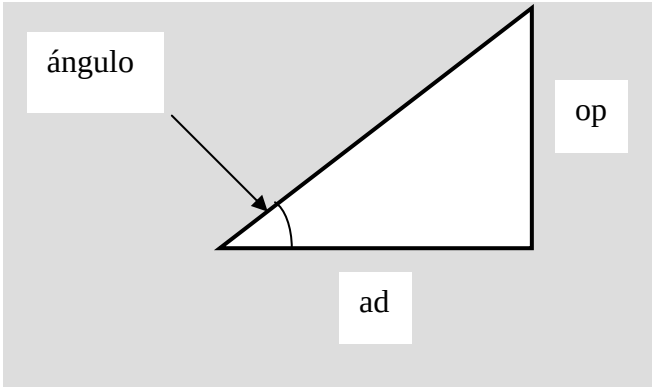
aquí aparentemente, las variables argumento (Re) y resultado (f) son las mismas en programa y función. Sin embargo no es así: como la función guarda las variables en su propio espacio de trabajo, la f de la función es una variable distinta de aquella del programa, y lo mismo pasa con Re. **Si bien es posible, y se admite que más intuitivo y natural usar el mismo nombre de variables para programa y función,** se recomienda no hacerlo, sobre todo cuando el programador es principiante, a efectos de enfatizar el hecho de que las variables se guardan en distintos lugares y por tanto son distintas, aunque puedan tener valores numéricos iguales.

Aunque es posible, no resulta conveniente modificar en una función las variables que se reciben como argumentos. Si las variables se llaman igual en programa y subprograma, Matlab sólo modificará la variable local en el subprograma, y su variable homónima del programa principal quedará sin modificar. De hecho, en las funciones propias de Matlab (y=sin(x),etc), el valor de *x* no se modifica en la función, sino que se usa como dato para calcular el valor de y.

se va a dar un ejemplo para el caso 2):

2.1)

Supóngase que un alumno tiene un ejercicio puramente trigonométrico donde debe calcular un cateto opuesto (op), dados el cateto adyacente (ad) y el ángulo considerado (ángulo)



la llamada podría ser

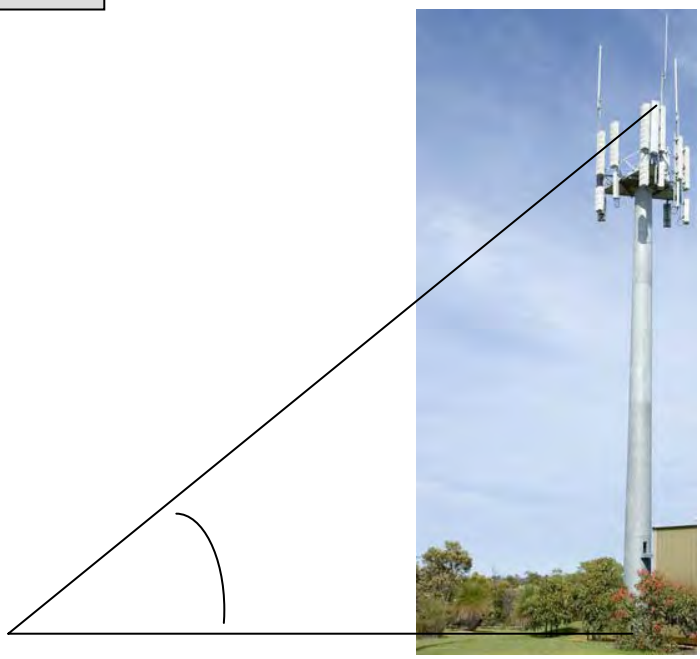
```
op =triang_rect(ad, angulo)
```

Mientras que la función podría escribirse así

```
function opuesto = triang_rect(adyacente, alfa)
% subprograma que calcula el cateto
opuesto, dados el adyacente y el ángulo

hipotenusa = adyacente./cos(alfa)

opuesto = hipotenusa.*sin(alfa)
```

2.2. Suponga ahora que, por triangulación, debe estimar la altura de una torre de telefonía inalámbrica, dada la distancia horizontal entre el observador y la misma, y el angulo de inclinación de la visión, que llamará beta.La función será la misma, pero la llamada podrá escribirse

```
alt_torre = triang_rect(dist_horiz, beta)
```



Variables globales

Una variable global es verdaderamente la misma en el programa llamador y en la función, ocupando un lugar en el espacio de trabajo (workspace). Siendo la misma, si se la modifica en la función, queda modificada inmediatamente en el espacio de trabajo para todos los módulos de programa que pueden acceder a ella. Por tanto, no se pasa como argumento a la función, sino que debe declararse en el programa llamador y en la función

Supóngase que el ángulo beta de triangulación para el cálculo de la torre se considera variable global. La llamada sería ahora

```
global beta

alt_torre = triang_rect(dist_horiz)
```

y la función ahora deberá escribirse así

```
function opuesto = triang_rect(adyacente)
% subprograma que calcula el cateto
opuesto, dados el adyacente y el angulo
global beta
hipotenusa = adyacente./cos(beta)

opuesto = hipotenusa.*sin(beta)
```

Si bien, como se ha visto, el uso de la declaración global parece simplificar la programación, su utilización constituye **una mala idea** puesto que expone a ciertas variables a una modificación no controlada en las funciones, que luego se mantienen modificadas al regresar el control al programa llamador. Esto puede resultar muy problemático cuando los programas son grandes.

Otro ejemplo de function.

Cálculo de un promedio y longitud de arreglo si la entrada es un vector

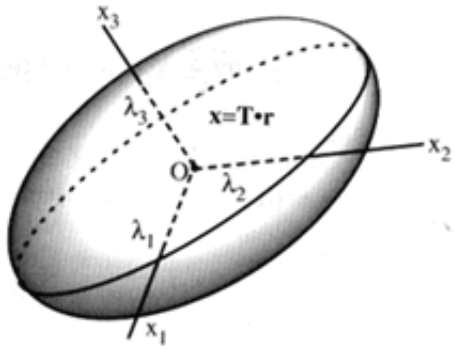
Se pueden ingresar estos comandos en un archivo M llamado prom.m. La function prom acepta un sólo arreglo como argumento de entrada y devuelve dos valores como argumentos de salida.

```
function [prome, long]=prom(variable)
 % prom, calcula promedio de vectores fila o columna
  [m,n] = size(variable);
if (m == 1 & n == 1)
    error('la entrada es un escalar');
else
    if (m==1|n==1)
        long = length(variable);
        prome = sum(variable)./long;
    else
        error('la entrada es una matriz');
    end
end
```

Se verá la operación de llamada desde el command window

```
>> clear
>> x = 1;
>> [promedio, longitud]=prom(x)
??? Error using ==> prom
la entrada es un escalar
>> clear
>> x = [1 1 2; 2 1 1; 3 1 2];
>> [promedio, longitud]=prom(x)
??? Error using ==> prom
la entrada es una matriz
>> x = 0:1:10;
>> [promedio, longitud]=prom(x)
promedio =
     5
longitud =
    11
```

Ejemplo de function que devuelve varios resultados por llamada
Cálculo de parámetros geométricos de un Elipsoide




```
%Programa principal
D= xlsread('dimens_elip');
l1= D(:1); l2= D(:,2); l3= D(:, 3);
n = length (1);
for i = 1:n
        % llama a funcion para calcular el volumen y el area superficial
        [vol_elip (i), area_sup(i), diameq(i), esf(i)]= elipsoide (l1(i),l2(i), l3(i))
end
plot (vol_elip)
figure(2)
plot (area_sup)
figure(3)
plot (diameq)
figure(4)
plot (esf)
```

```
function [ve, ase, de, fe] = elipsoide (long1, long2, long3)
 % Calcula volumen del elipsoide
ve = pi./6.*long1.*long2.*long3;
% Calcula area del elipsoide
longmed = (long2+long3)./2;
U = sqrt (long1.^2-longmed.^2)./long1;
ase = pi./2.*long1.*longm.*(longm./long1 + 1./U.*asin(U))
% calcula el diametro equivalente
de = (6.*ve./pi).^(1./3)
% calcula factor de esfericidad
fe = pi.*de.^2./ase
```

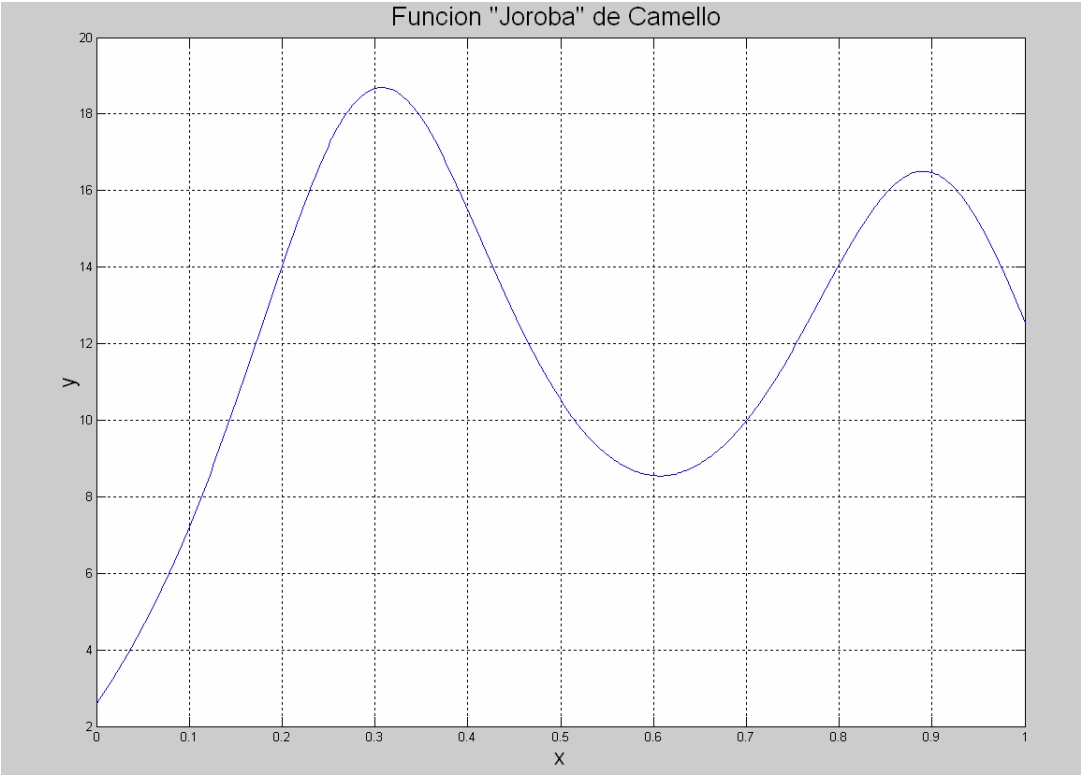
Funciones de Función

Existe un tipo de funciones, generalmente preprogramadas,que opera con funciones como argumentos, denominadas "función de funciones" Las funciones de función incluyen:

Búsqueda de raíces
Optimización
Cuadratura (Determinación numérica del área bajo la curva de una Integral definida)
Ecuaciones Diferenciales Ordinarias

MATLAB representa la función no lineal mediante una function de archivo-M. Por ejemplo, aquí tenemos una versión simplificada de la función giba del directorio matlab/demos .

```
clear
x = 0:0.002:1;
y = giba(x);
plot (x,y)
grid
xlabel ('x', 'Fontsize', 18)
ylabel ('y', 'Fontsize', 18)
Title ('Funcion "Joroba" de Camello ', 'Fontsize', 20)

function ord = giba(absc)
ord = 1./((absc-.3).^2 + .045) + 1./((absc-.9).^2 + .05) - 6;
```



El gráfico muestra que la función tiene un mínimo local cercano a *x* = 0.6. La función fminsearch encuentra el *minimizador*, que es el valor de *x* donde la función adquiere su valor mínimo. El primer argumento para fminsearch es un "function handle" (@) o "manipulador" para la función que está siendo minimizada. Esto permite utilizarla como argumento de otra función y el segundo argumento es una estimación de la posición del mínimo.

p= fminsearch(@giba, 0.5)
p =
0.6067
Para evaluar la función en el minimizador,
giba(p)
ans =
8.5425

En análisis numérico, se utilizan los términos *cuadratura* e *integración* para distinguir entre la valuación numérica de integrales definidas (area bajo la curva) y la integración numérica de ecuaciones diferenciales ordinarias. Las rutinas MATLAB de cuadratura son quad (Método de Simpson adaptativo) y quadl (Método adaptativo de Lobatto).

La instrucción

Q = quad (@giba,0,1)
Calcula el área bajo la curva en el gráfico y produce
Q =
12.3376

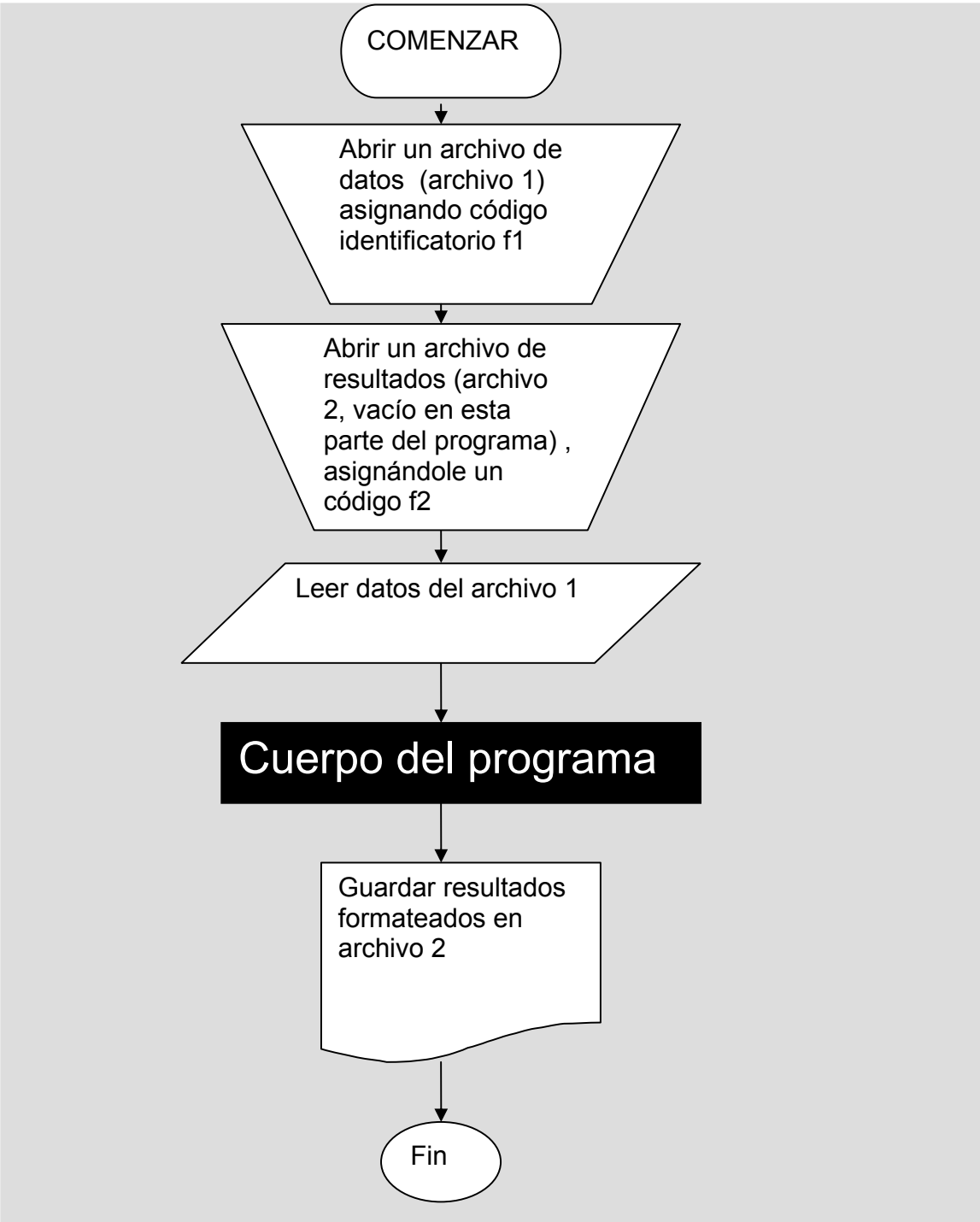
Finalmente el gráfico muestra que la función nunca se anula en este intervalo. Así, si Ud. buscara una raíz en él, con una estimación inicial en el centro del intervalo

raiz = fzero(@giba, 0.5)

encontrará el resultado afuera del intervalo
z =
-0.0920

CAPITULO 8. INSTRUCCIONES DE ENTRADA Y SALIDA CON FORMATOS.

Diagrama conceptual del uso de archivos de entrada o salida con formato



APERTURA DE ARCHIVOS
Asignación de codigo identificador

FOPEN

FID = FOPEN(Nombre de archivo) abre el archivo FILENAME para acceso a lectura.

Si el archivo se abre para lectura y no se lo encuentra en el directorio de trabajo, FOPEN busca en la ruta de búsqueda (search path) de MATLAB.

FID es un escalar entero de MATLAB denominado, "identificador de archivo". Se puede usar fid como primer argumento para otras rutinas de entrada/salida. Si FOPEN no puede abrir el archivo, devuelve -1.

FID = FOPEN(Nombre de archivo, PERMISO) abre el archivo "Nombre de archivo" en el modo especificado por PERMISO. PERMISO puede ser:

'r' read (lectura, archivo previamente existente)
'w' write (Escritura, lo crea si es necesario)
'a' append (agregar datos a un archivo existente(si no existe lo crea))

Existen dos identificadores de archivo que están disponibles automáticamente y no necesitan ser abiertos. Ellos son FID=1 (salida normal) y FID=2 (error normal).

[FID, MENSAJE] = FOPEN(NOMBREDEARCHIVO, PERMISO) devuelve un mensaje de error dependiente del sistema si la apertura del archivo no es exitosa.

fprintf

Escribe datos formateados a un archivo

Sintaxis

count = fprintf(fid,format,A,...)

Imprime resultados formateados en la parte real de la matriz A (y de cualquier argumento adicional) bajo el control de una cadena de conversores de formato. Estos resultados se escriben a un archivo identificado con el código fid.

fprintf devuelve un recuento del número de bytes escrito a la variable "count". (**ESTA PUEDE NO USARSE)**

El argumento fid es un identificador entero obtenido con un fopen. La omisión de fid envía la salida a la pantalla.

Ejemplo 1. Como imprime naturalmente el fprintf?

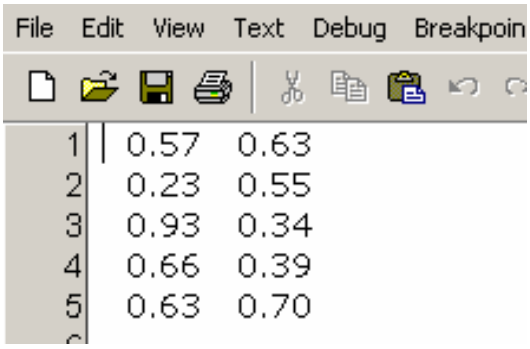
Supóngase que se genera una matriz de números aleatorios de 5 x 2, es decir, de 5 filas por dos columnas. Sabiendo que **fprintf** imprime la matriz entera, vamos a ver el la disposición que establece en el archivo de salida si el formato permite escribir dos valores por línea. Escribirá las filas correctamente?

```
clear, clc
fid = fopen('salen.txt', 'w')
MATIMP = rand (5,2) % 5 filas x 2 columnas
fprintf (fid, '%6.2f % 6.2f\n', MATIMP)
fclose(fid)
```

Al ejectuar el programa (por ejemplo "efeprintefe.m"), el command window muestra lo siguiente:

```
>>fid =
     5
MATIMP =
    0.5692    0.3352
    0.6318    0.6555
    0.2344    0.3919
    0.5488    0.6273
    0.9316    0.6991
```

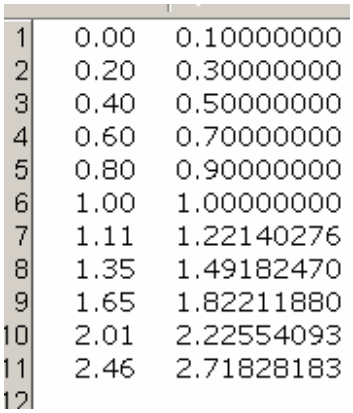
es decir, la matriz se visualizada por no suprimir la salida, se presenta como uno lo esperaba, de 5 filas x 2 columnas. Se verá que sucede con el archivo "salen.txt", que es lo que nos importa aquí



se observa que la primera fila de los resultados en este archivo son los elementos MATIMP(1,1) y MATIMP (2,1), y que la segunda fila está compuesta por MATIMP (3,1) y MATIMP (4,1). La tercera fila está compuesta por los elementos MATIMP(5,1) Y MATIMP (1,2), porque recurre ya a la segunda columna. Las filas que siguen en el archivo visualizado están formadas sólo por elementos de la segunda columna originales, MATIMP (2,2), MATIMP (3,2) (cuarta fila del archivo), y MATIMP (4,2) y MATIMP (5,2) (quinta fila). **En consecuencia, la matriz se imprime "por columna".** Estas instrucciones de Matlab para la impresión con formatos provienen, del lenguaje C. La especificación de formato 6.2f implica campos de 6 posiciones totales, y entre ellas el punto decimal y dos dígitos decimales. Esto permite, en este caso, que sobren 2 espacios a la izquierda de cada campo. La precisión decimal sale por redondeo.

Ejemplo 2. ¿Como imprimir una Tabla x-y?
2.1. Forma incorrecta
En este programa no se leen datos, se genera un vector de valores de "x", se calcula la variable dependiente "y", se abre un archivo para guardar los resultados, y se graban los mismos con una determinada especificación de formato. Supóngase que se quiere imprimir una tabla x,y.

```
x = 0:.1:1;  % vector fila 1 x 11
y = exp(x);  % vector fila 1 x 11
fid=fopen('salidaxy.txt','w');
fprintf(fid, ' %6.2f  %12.8f\n',x,y);
fclose(fid);
```



1	0.00	0.10000000
2	0.20	0.30000000
3	0.40	0.50000000
4	0.60	0.70000000
5	0.80	0.90000000
6	1.00	1.00000000
7	1.11	1.22140276
8	1.35	1.49182470
9	1.65	1.82211880
10	2.01	2.22554093
11	2.46	2.71828183
12		

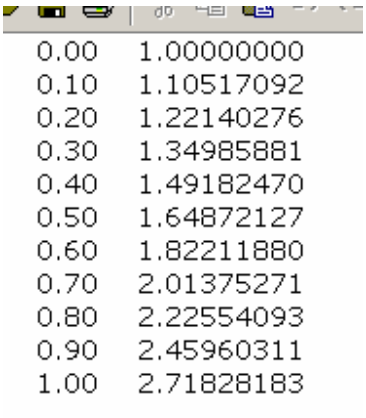
-La instrucción fprintf "imprime" (visualiza) en el archivo "salidaxy.txt" todos los valores del arreglo x primero, de a dos por línea, y luego todos los de y, de a dos por línea también. El carácter \n es lo que permite pasar a la línea siguiente (linefeed). De todas maneras, esto no permite obtener la tabla buscada.

2.2. Forma correcta
Una manera de lograrlo sería mediante la construcción de una matriz bidimensional, donde la primera fila contuviera los valores de x, y la segunda, los de y. Si al **fprintf** se le indica escribir dos valores por línea, Matlab buscaría los datos por columna y escribiría un valor de **x** y otro de **y** por fila del archivo de resultados. Es decir, dado que el número de variables a visualizar por línea (por fila del archivo de resultados), coincide con el número de columnas de la matriz original, lo que haría fprintf es trasponer la matriz y presentarla en el archivo.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1.11	1.22	1.35	1.49	1.65	1.82	2.01	2.23	2.46	2.72

* (los valores se escribieron en esta tabla con dos decimales sólo para ilustrar la disposición de los números y no su precisión)

```
x = 0:.1:1; % vector fila 1 x 11 col
y = exp(x); % vector fila 1 x 11 col
IMP = [x;y] % matriz de 2 filas x 11 col
fid=fopen('salidamat.txt','w');
fprintf(fid, ' %6.2f  %12.8f\n',IMP );
fclose(fid);
```



0.00	1.00000000
0.10	1.10517092
0.20	1.22140276
0.30	1.34985881
0.40	1.49182470
0.50	1.64872127
0.60	1.82211880
0.70	2.01375271
0.80	2.22554093
0.90	2.45960311
1.00	2.71828183

Como se observa en el archivo "salidamat.txt" ahora sí se obtiene el tipo de salida que esperado.

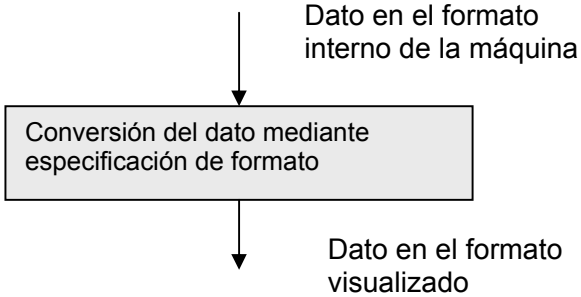
-La instrucción **fopen** asigna un código **fid** al archivo **salidamat.txt** que se utiliza para escribir (w por write) resultados. La instrucción fclose cierra el archivo.

-Cada elemento de la primera columna se visualiza con especificación de conversión ("conversor") real "f" de 6 espacios totales, de los cuales 2 son decimales

-Cada número de la segunda columna se visualiza con un conversor real "f" de 12 espacios totales, de los cuales 8 son decimales.

-Esta conversión 12.8f excede lo necesario para la segunda columna y como por defecto MATLAB justifica el valor a la derecha del campo, parte del ancho de campo se emplea para separar sus valores de aquellos de la primera columna.

Argumento de Formato Es una cadena de caracteres con conversores de formato.



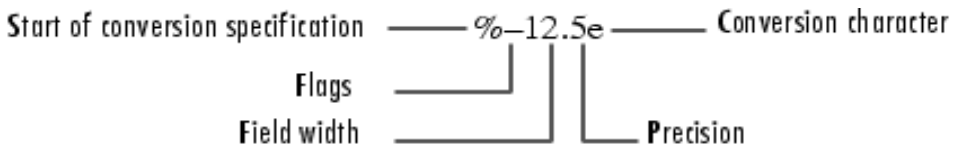
Los datos se almacenan internamente con muchos decimales (cada dato ocupa normalmente 8 bytes), muchos de los cuales exceden la precisión verdadera de variables físicas. Así, uno puede querer expresar el dato con una determinada cantidad de decimales, de allí es que puede surgir la necesidad de convertirlos.

Un conversor de formato controla la notación, alineación, dígitos significativos, ancho de campo, y otros aspectos del formato de salida.
-La cadena de formato puede contener caracteres de escape para representar caracteres no impresos como los caracteres de línea nueva y tabuladores.

Las especificaciones de conversión comienzan con un carácter % y contienen estos elementos opcionales y requeridos

- Flags (carteles identificatorios, opcional)
- Ancho de campo (Field width) y precisión (opcional)
- Un especificador de subtipo (opcional)
- Un carácter de conversión (requerido)

Usted puede especificar estos elementos en el orden siguiente



Comienzo de la especificación de conversión, indicador, ancho de campo, precisión, carácter de conversión

Flags (indicador) Se puede controlar la alineación de la salida usando cualquiera de estos indicadores opcionales.

Character	Description	Ejemplo
Un signo menos (-)	Justifica a la izquierda el argumento convertido en este campo	%-5.2d
Un signo más (+)	Siempre imprime el signo (+ o -)	%+5.2d
Cero (0)	Incorpora ceros en lugar de espacios	%05.2d

Especificación de ancho de campo y precisión Se puede controlar el ancho y la precisión de la salida incluyendo estas opciones en la cadena del format

Carácter	Descripción	Ejemplo
Ancho de campo	Una cadena de dígitos que especifica el número mínimo de dígitos a ser impreso	%6f
Precisión	Una cadena de dígitos con (.) que indica el número de dígitos impreso a la derecha del punto decimal	%6.2f

Caracteres de Conversión: especifican la notación de la salida

Especificador	Descripción
%c	Carácter individual
%d	Notación decimal entera (con signo)
%e	Notación exponencial (e significa 10 y lo que está a la derecha, el exponente) (usando minúsculas e como en 3.1415e+00)
%E	Notación exponencial (usando mayúsculas E como en 3.1415E+00)
%f	Notación de punto flotante
%g	Versión compacta de %e o %f . Los ceros no significativos no se imprimen.
%G	Igual que %g, pero usando mayúscula E
%s	Cadena de caracteres

***Caracteres de Escape*:** Esta tabla lista la secuencia de caracteres de escape, es decir, caracteres no imprimibles de un formato.

Carácter	**Descripción**
\b	Backspace (retroceso)
\f	Alimenta página
\n	**Nueva línea**
\r	Retorno de carro
\t	Tabulación horizontal
\\	Barra invertida
\" or '' (Dos apostrofes)	Indicador de carácter dentro de la especificación de formato
%%	Carácter de porcentaje

Impresión de una constante en pantalla

fprintf('un círculo unitario tiene una circunferencia %g radianes.\n',2*pi)

muestra una línea sobre la pantalla

Un círculo unitario tiene una circunferencia 6.283186 radianes.

Ejemplo elemental de tabla, con mensaje de error

```
% abrir el archivo
 fid = fopen ('miarchivo.txt', 'w');
 if (fid < 0)
    error ('no pude abrir el archivo "miarchivo.txt" ');
end
 % escriba algo al archivo
fprintf(fid, 'number     Square\n')
for (i = 1:10)
    fprintf(fid, '%3d %6d\n', i,i.^2);
end
%cerrar el archivo
fclose (fid);
```

	number	Square
1		
2	1	1
3	2	4
4	3	9
5	4	16
6	5	25
7	6	36
8	7	49
9	8	64
10	9	81
11	10	100
12		

Tablas con fprintf y for

Tabla x-y

Supóngase que para el ejemplo inicial, en lugar de crear la matriz de impresión IMP, deseamos imprimir los vectores x e y, en forma de tabla, es decir

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1.11	1.22	1.35	1.49	1.65	1.82	2.01	2.23	2.46	2.72

Como se ha mencionado dentro del archivo y en la parte inicial de esta clase, MATLAB imprime un arreglo completo, y luego otro. Para evitar esto, se introduce la sentencia fprintf en un lazo repetitivo **for** que la ejecuta tantas veces como elementos tenga el vector, imprimiendo un elemento de cada arreglo por vez. Podemos decir que del x(i) pasa al y(i) porque no tiene más que un valor de cada arreglo cada vez que imprime.

```
x = 0:.1:1; y = exp(x);

n= length(y); fid=fopen('salidaxyfor.txt','w');

for i = 1:n

   fprintf(fid,'%6.2f %12.8f\n',x(i), y(i));

end

fclose(fid);
```

Problema de la ecuación cuadrática

Se recordará que en un ejemplo anterior, se resolvieron una ecuación cuadrática $ax^2 + bx + c = 0$ para n ternas de valores a, b, y c, generando dos raíces por terna, cada una con una componente real y otra imaginaria, esto es, los vectores x1r, x1i, x2r y x2i, cada uno de n valores.

Si se deseara construir una tabla del tipo

x1r	x1i	x2r	x2i
----	----	----	----
----	----	----	----
----	----	----	----
.	.	.	.
.	.	.	.
.	.	.	.
----	----	----	----

se puede escribir el segmento de programa siguiente, asumiendo que ya están definidos los vectores de raíces

```
% se abre el archivo de salida "salidaraices.txt"
fid = fopen('salidaraices.txt','w');
%se imprime el encabezamiento de la tabla (cadena de caracteres)
%una sola vez
fprintf(fid, 'x1r              x1i              x2r              x2i\n')
% se imprimen los valores del vector. Al tener cada vez Matlab un elemento
%de cada arreglo, pasa al otro, y al otro hasta escribirlos todos en una línea
for k = 1:n
  fprintf (fid, '%8.4f        %8.4f        %8.4f        %8.4f\n', x1r(k), x1i(k), x2r(k), x2i(k) )
end
```

LEER DATOS DESDE PLANILLAS DE CALCULO EXCEL, DESDE UN PROGRAMA
xlsread
Lee planillas de Microsoft Excel (.xls)
Sintaxis
A = xlsread('nombre de archivo')

devuelve datos numéricos en el arreglo A a partir de la primera página en la planilla Microsoft Excel denominada 'Nombre de archivo'.

xlsread ignora los encabezamientos de texto en filas o columnas.
Pero si una celda no es cabeza de fila o columna, está vacía o contiene texto, xlsread coloca un NaN en esa posición de A.
Como traer los encabezamientos de las columnas de la planilla Excel a MATLAB
[A, B] = xlsread('nombre de archivo')
devuelve datos en un arreglo numérico A, y datos de texto en un arreglo de celdas B.
Si la planilla contiene encabezamientos de filas o columnas con texto, xlsread devuelve solo esas celdas en B.

Ejemplos
Ejemplo 1 – Leyendo datos numéricos
El archivo de planilla de cálculo Microsoft, testdata1.xls, contiene estos datos:

1	6
2	7
3	8
4	9
5	10

Para leer este dato en MATLAB, utilice este comando:

```
A = xlsread('testdata1.xls')
A =
   1   6
   2   7
   3   8
   4   9
   5  10
```

Ejemplo 2 – Manejo de datos de texto
La planilla testdatos2.xls, contiene datos numéricos y de texto.

1	6
2	7
3	8
4	9
5	Text

la sentencia xlsread inserta un “NaN” (Not-a-number) en lugar del texto en el resultado.

```
A = xlsread('testdata2.xls')
A =
   1   6
   2   7
   3   8
   4   9
   5 NaN
```

Ejemplo 3 – Manejo de archivos con encabezamientos de filas y columnas
La planilla tempdata.xls, contiene dos columnas de datos numéricos con encabezamiento de texto para cada columna:

Tiempo	Temperatura
12	98
13	99
14	97

Si se desea solamente importar los datos numéricos, utilice xlsread con un sólo argumento de retorno. Así, xlsread ignorará las celdas de texto en el resultado numérico.

```
ndatos = xlsread('tempdata.xls')
ndatos =
   12   98
   13   99
   14   97
```

Para importar tanto los datos numéricos como los de texto, se especifican dos datos de retonro para xlsread.

```
[ndatos, encabezamientos] = xlsread('tempdata.xls')
ndatos =
   12   98
   13   99
   14   97
encabezamientos =
   'tiempo'    'temperatura'
```

Leyendo archivos ASCII de texto con formato

fscanf

Lee datos formateados desde un archivo de datos ASCII

Sintaxis

A = fscanf(fid,formato)

[A,count] = fscanf(fid,formato, tamaño)

Descripción

A = fscanf(fid,formato) lee todos los datos de un archivo especificado por fid, convirtiéndolo de acuerdo a una cadena especificadora de formato, y asignándolo a la matriz A. El argumento fid es un archivo identificador entero, obtenido mediante una instrucción fopen. “Formato”, es una cadena de caracteres que especifica el formato de los datos a ser leídos

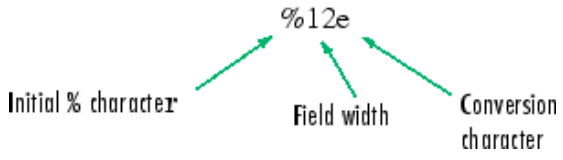
[A,count] = fscanf(fid,formato,tamaño) lee los datos especificados por “tamaño”, y los convierte de acuerdo a la cadena de caracteres de formato y los asigna a una matriz, junto al recuento de elementos leídos existosamente (count). El argumento “size”, determina la cantidad de datos que deben leerse. Opciones válidas para tamaño, son

n	**Lee hasta n números, caracteres o cadenas de caracteres.**
inf	**Lee hasta el final del archivo.**
[m,n]	**Lee hasta(m*n) números, caracteres o cadenas de caracteres. Llena, en orden de columna, una matriz de hasta m filas. El símbolo n puede ser inf, pero m no.**

como se ha dicho, fscanf es una instrucción vectorizada, de manera que la cadena de formato se usa una y otra vez hasta encontrar un final de archivo, o se alcanza la cantidad de datos a leer especificada por “tamaño”.

Comentarios

Cuando MATLAB lee un archivo especificado, intenta establecer la correspondencia entre los datos del archivo y la cadena de formato.
Si se da tal situación, los datos se escriben en la matriz en forma de columna. Si se da una correspondencia parcial, sólo los datos que coincidan se escriben en la matriz, y la operación de lectura se detiene.
La cadena de formato consiste en caracteres comunes y especificaciones de conversión. Estas últimas indican el tipo de datos que debe corresponderse con los del archivo leído e involucran el carácter %, campos de ancho opcional y caracteres de conversión, organizados como se indica abajo:



Carácter inicial, indicador, ancho de campo carácter de conversión

Obsérvese que no es necesario establecer precisión al leer con fscanf

Ejemplo: supongamos que tenemos una matriz de 4 x 3, por ejemplo, generada con una función randomize.

```
clear, clc
A = rand(4,3)
%se asigna un identificador al archivo donde se va a escribir la matriz A
fi = fopen ('mifichero.txt', 'w')
% Matlab recorre la matriz A por columna
% En la instruccion fprintf se indican los valores que se despliegan por fila del archivo
% Para guardar la matriz original, hay que generar la transpuesta
At = A'
% y escribir la transpuesta, de 3 filas por 4 columnas. Es decir, cada columna tiene
%tres valores si se le indica a Matlab que despliegue tres valores por linea, va a
%escribir las columnas de la transpuesta como filas, recuperando en el archivo la
%matriz original

fprintf(fi, '%8.5f    %8.5f    %8.5f\n', At)
fclose(fi)
% se abre el archivo para lectura
fm = fopen ('mifichero.txt', 'r')
%como fscanf va a recorrer el archivo por filas y lo asigna a una matriz por columnas,
se le va a decir que la matriz que asigne tenga tres filas.Ese mecanismo va a volver
a generar la transpuesta de la matriz A, llamada Afs
Afs=fscanf(fm, '%f',[3,Inf])
% el 3 corresponde al numero de valores que tendrá cada columna”
% se arma nuevamente la matriz, que se llama ahora "matriz recuperada", Ar
Ar = Afs'
%finalmente, se va a leer el archivo directamente para generar un vector columna
fv = fopen('mifichero.txt','r')
Av=fscanf(fv, '%f ', Inf)
```

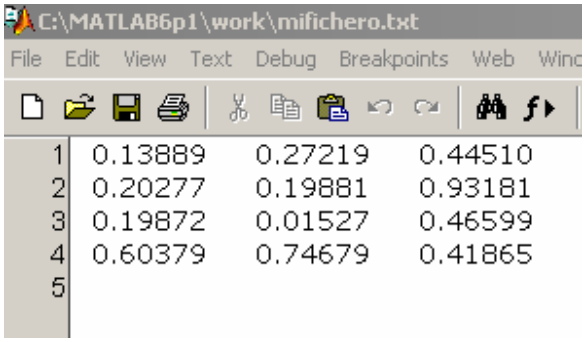
Esta sería la visualización del command window

```
>> A =

    0.1389    0.2722    0.4451
    0.2028    0.1988    0.9318
    0.1987    0.0153    0.4660
    0.6038    0.7468    0.4186
```

Se genera la traspuesta

```
At =

    0.1389    0.2028    0.1987    0.6038
    0.2722    0.1988    0.0153    0.7468
    0.4451    0.9318    0.4660    0.4186
```

fprintf va a recorrer por columna y desplegar los valores en filas de tres valores



que nos devuelve la matriz original, guardada en “mifichero.txt”
A veces es muy útil utilizar la instrucción fscanf para leer de archivos que fueron creados previamente con fprintf

Afs=fscanf(fm, '%f',[3,Inf]) % el 3 corresponde al numero de valores que tendrá cada columna, es decir, el número de filas

```
Afs =

    0.1389    0.2028    0.1987    0.6038
    0.2722    0.1988    0.0153    0.7468
    0.4451    0.9318    0.4660    0.4187
```

Es decir, nos daría la traspuesta del contenido del archivo. Entonces, trasponiéndola, se recupera la matriz original

```
Ar =
    0.1389    0.2722    0.4451
    0.2028    0.1988    0.9318
    0.1987    0.0153    0.4660
    0.6038    0.7468    0.4187
```

A su vez, si dejamos que fscanf asigne sin indicarle más que el carácter de conversión, se tendrá un vector columna

```
Av =

    0.1389
    0.2722
    0.4451
    0.2028
    0.1988
    0.9318
    0.1987
    0.0153
    0.4660
    0.6038
    0.7468
    0.4187
```

CAPITULO 9. VECTORIZACION DE CÓDIGO EN MATLAB. Sandro M. Goñi

En MATLAB los lazos repetitivos pueden ser reemplazados por estructuras pseudosecuenciales (vectorizadas), siempre y cuando se pueda encontrar o desarrollar una expresión vectorizada equivalente (y simple!) que resuelva satisfactoriamente el problema particular. Algunas razones para realizar la vectorización:

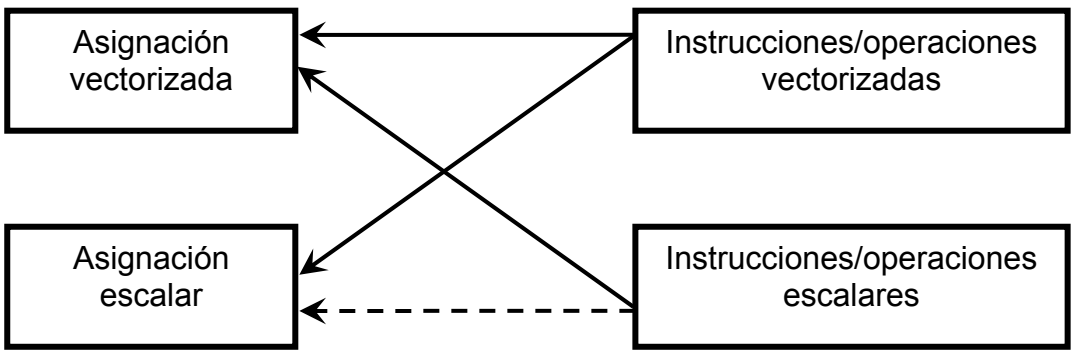
- Programación mas simple y concisa.
- Menos tiempo de procesamiento.
- Indexación automática.

En estas notas se dan algunos ejemplos de vectorización. Se sugiere tipearlos y reproducir los resultados, observando la respuesta de cada instrucción.

Como punto de partida, la vectorización consiste básicamente en la sentencia o instrucción de operaciones, en forma directa o con un solo comando simple (o una serie de ellos), sobre variables típicamente no escalares (vectores y matrices), para dar un resultado escalar o no escalar. La vectorización es, en este sentido, una estructura "pseudosecuencial", que puede involucrar una gran cantidad de operaciones simples, las cuales son indicadas por una instrucción aparentemente sencillas. Es importante destacar que durante las primeras clases introductorias a MATLAB se utilizaron repetidamente estructuras vectorizadas; por ejemplo, las instrucciones sencillas para graficar la función seno:

```
x=0:pi/10:2*pi;
y=sin(x);
plot(x,y)
```

encierran operaciones y asignaciones vectorizadas. Este esquema muestra los posibles caminos de las formas vectorizadas y de las formas escalares tradicionales.



No vectorizado (- - - - - - - ▸) / Vectorizado (——————▸)

Se mostraran diversos ejemplos de programación "escalar" y formas vectorizadas equivalentes. Un ejemplo simple, consiste en generar los elementos al cuadrado del vector x=[0 1 2]. La generación del vector, componente a componente, puede realizarse como:

```
...
N=length(x);
for i=1:N
    y(i)=x(i)^2;
end
...
```

Aquí la operación se realiza para cada componente del vector x en forma individual y sucesiva, con control de posición ejercido por la variable de control del lazo for. Dicho de otra manera, la operación se realiza en forma "escalar". Al mismo tiempo, la asignación es tambien escalar, controlada por el lazo for.

En forma vectorizada, las operaciones se indican como:

```
y=x.^2;
```

Donde el operador de arreglos ".^" es utilizado para indicar la potenciación elemento a elemento. De esta manera las operaciones se "programan" en forma vectorizada, y el resultado de la misma es tambien vectorizado.

Suponga que se quiere calcular el producto escalar entre los vectores x=[1 5 2] e y=[-1 0 8]. En forma escalar se puede escribir como:

```
s=0;
for i=1:length(x)
    s=s+x(i)*y(i);
end
```

En forma vectorizada:

```
s=x*y'
```

En este caso, la operación se realiza en forma vectorizada (* y no .*), mientras que el resultado es escalar.

Suponga ahora que se quiere generar el vector z=[1 1 1]. En forma escalar se puede escribir:

```
for i=1:3
    z(i)=1;
end
```

Mientras que en forma vectorizada se puede escribir:

```
z(1:3)=1
```

En este caso la operación (asignación directa) es escalar, mientras que el resultado se asigna en forma vectorizada.

Ejemplo 1: reemplazar las componentes negativas de una matriz genérica con 0.

Con lazo for e if para encontrar valores negativos:

```
for i=1:size(A,1) % le pedimos a size el número de filas
    for j=1:size(A,2) % ahora le pedimos el número de columnas
        if A(i,j)<0
            A(i,j)=0;
        end
    end
end
```

Indexación lógica

A(B), donde B es un arreglo de valores lógicos, devuelve los valores de A para aquellos índices donde la parte real de B es distinta de cero (B debe tener el mismo tamaño que A). A(B) es equivalente a la expresión A(find(B)) (vea *help logical indexing*)

Vectorizado:
Aplicando entonces la indexación lógica, tendremos:

```
A(A<0)=0 % que es equivalente a:
A(find(A<0))=0
```

El resultado de la expresión A<0 es una matriz de ceros y unos del mismo tamaño de A, con componentes iguales a "1" en las posiciones o índices donde se cumpla la condición. Para esas posiciones, los elementos de la matriz se hacen igual a "0".

Ejemplo 2: reemplazar las componentes menores de un umbral mínimo de una matriz genérica con un valor mínimo prefijado, y las componentes mayores de un umbral máximo con un valor máximo prefijado.

Con lazo for e if para encontrar las restricciones:

```
vmin=2;
vmax=10;
for i=1:size(A,1)
    for j=1:size(A,2)
        if A(i,j)<vmin
            A(i,j)=vmin;
        elseif A(i,j)>vmax
            A(i,j)=vmax;
        end
    end
end
```

Vectorizado, utilizando operadores relacionales:

```
A(A<vmin)=vmin;
A(A>vmax)=vmax;
```

Ejemplo 3: Calcular el valor de la función f(x), para un vector x arbitrario, con f(x) definida como:

$$f(x)=\begin{cases} x^2+1 & x<0 \\ -x+2 & 0\le x<20 \\ \dfrac{x^2}{x-1} & x\ge 20 \end{cases}$$

Con lazo for e if:

```
x=randn(1,100)*50;
for i=1:length(x)
    if x(i)<0
        f(i)=x(i)^2+1;
    elseif x(i)>=0&x(i)<20
        f(i)=2-x(i);
    else
        f(i)=x(i)^2/(x(i)-1);
    end
end
```

Vectorizado:

```
x=randn(1,100)*50;
f=(x.^2+1).*(x<0)+(2-x).*(x>=0).*(x<20)+(x.^2./(x-1)).*(x>=20);
```

¿Puede explicar como funciona esta asignación?

Ejemplo 4: Generar la matriz de Hilbert de tamaño *NxM* (la misma se define como A(i,j)=1/(i+j-1)).

Con lazo for:

```
N=5;
M=8;
for i=1:N
    for j=1:M
        A(i,j)=1/(i+j-1);
    end
end
```

Vectorizado:

```
N=5;
M=8;
A=1./([1:N]'*ones(1,M)+ones(N,1)*[1:M]-1)
```

Explicación: [1:N]'*ones(1,M) genera la matriz de índices fila:

```
   1   1   1   1   1   1   1   1
   2   2   2   2   2   2   2   2
   3   3   3   3   3   3   3   3
   4   4   4   4   4   4   4   4
   5   5   5   5   5   5   5   5
```

y ones(N,1)*[1:M] genera la matriz de índices columna:

```
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
   1   2   3   4   5   6   7   8
```

Así la suma de ambas matrices debuelve la suma de indices de cada elemento de la matriz:

```
 2   3   4   5   6   7   8   9
 3   4   5   6   7   8   9  10
 4   5   6   7   8   9  10  11
 5   6   7   8   9  10  11  12
 6   7   8   9  10  11  12  13
```

La asignación de A es directa, dividiendo 1 (componente a componente) por la suma de matrices índices-1.

Hasta este punto, podemos observar que utilizando lazo for:
- hay que utilizar índices para indicar sobre que componente se esta trabajando
- se deben utilizar índices para indicar a que componente de la variable debe asignarse el resultado de la operación (si es necesario)
- se necesitan mayor cantidad de escritura

El uso de la forma vectorizada es más directo, con mayor o menor dificultad para encontrar una forma adecuada de expresar la misma.

En ocaciones se pueden vectorizar algunas partes de las operaciones, dentro de un lazo principal exterior

Ejemplo 5: generar una matriz tridiagonal de *NxN* donde la diagonal principal tenga valores igual a 2, y las diagonales secundarias valores iguales a -1.

Con lazo for e if:
```
N=10;
for i=1:N
    for j=1:N
        if i==j
            A(i,j)=2;
        elseif i+1==j
            A(i,j)=-1;
        elseif j+1==i
            A(i,j)=-1;
        end
    end
end
```

Con lazo for (aprovechando que es tridiagonal):
```
N=10;
A(1,[1 2])=[2 -1]
for i=2:N-1
    A(i,[i-1 i i+1])=[-1 2 -1];
end
A(N,[N-1 N])=[-1 2]
```

Note que la instrucción dentro del lazo esta vectorizada.

Vectorizado 1,utilizando la función diag:
```
N=10;
A=diag(ones(1,N))*2-diag(ones(1,N-1),1)-diag(ones(1,N-1),-1)
```

CAPÍTULO 10. DIFICULTADES Y ERRORES TÍPICOS

Ing. Sandro M. Goñi

A continuación se enumeran algunas de las dificultades encontradas y errores que los alumnos cometen más frecuentemente durante la práctica. Si bien existen muchas otras situaciones (casi tanto como imaginemos), aquí se nombran los mas importantes y habituales.

1) Errores de indexación

Estel tipo de error es importante y uno de los más ampliamente observados. Debe recordar que la indexación **siempre** se realiza con números naturales (excepto indexación lógica), pero a pesar de esta muy clara definición, muchas veces se insiste con utilizar variables con valores no naturales como índices.

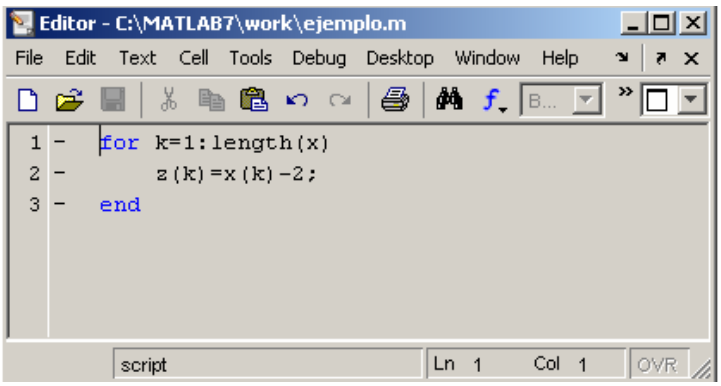
Por ejemplo:

```
>> x=rand(1,5)

x =

    0.9501    0.2311    0.6068    0.4860    0.8913

>> x(0)
??? Subscript indices must either be real positive integers or logicals.

>> x(1.19)
??? Subscript indices must either be real positive integers or logicals.
```

Otra situación observada es la indexación de escalares, lo que naturalmente conduce a errores.

2) Definición de Variables y funciones

Es también uno de los errores más comunes, utilizar una variable o función que no se ha creado. Por ejemplo:



Se obtiene:

```
??? Undefined function or variable "x".

Error in ==> ejemplo at 1
for k=1:length(x)
```

dado que x no esta definida. Otro error común es utilizar una variable antes de definirla:

```
T=0:DT:100;
```

```
DT=10;
```

Además, muchos comandos como clear, global, etc., pueden ser utilizados incorrectamente, pudiendo producir errores posteriores. Los argumentos deben escribirse separados por espacios, no por comas. Si se utiliza una "coma" en la lista, lo que viene luego de la misma se interpreta como una sentencia separada.
Por ejemplo, si declara a y b globales, la forma de hacerlo es:

```
global a b
```

y no:

```
global a,b
```

en cuyo caso b no es global, y simplemente se muestra su valor. Además la declaración de global se realiza antes de crear y asignar un valor a la variable, por lo cual se obtendrá un error, dado que no puede mostrarse b por que no esta definida.

3) Errores de sintaxis

Un error de sintaxis puede ser, por ejemplo:

```
for k=1#10
    z(k)=2*k^2;
end
```

En este caso, MATLAB responderá con:

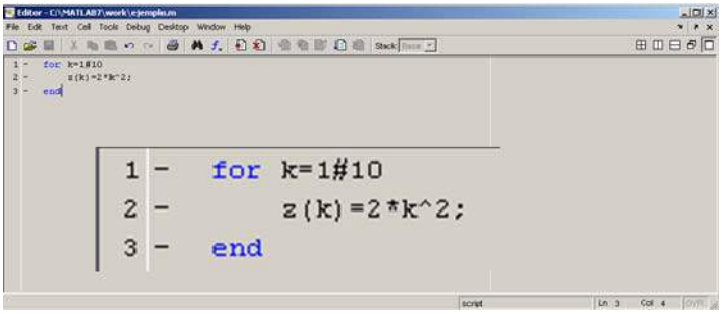
```
??? for k=1#10
            |
Error:  The  input  character  is  not  valid  in  MATLAB  statements  or
expressions.
```

ó (dependiendo de la versión):

```
??? for k=1#10
            |
Error: Missing variable or function.
```

Aquí se utilizó "#" en lugar de ":" (*colon*) y se produce un mensaje de error indicando el problema.

Cuando se comete una equivocación en un script, se obtiene un mensaje de error indicando el nombre del archivo, la línea y columna donde se produce el error, y el tipo de error.




```
??? Error: File: C:\MATLAB7\work\ejemplo.m Line: 1 Column: 8
```

```
Missing variable or function.
```

4) Errores de sintaxis que no producen errores

Un error de sintaxis que no se detecta, por ejemplo:

```
for k=1,10
    z(k)=2*k^2;
end
```

Aqui se obtendrá un vector z de longitud 1, y luego se obtendrá ans=10. Este tipo de error es completa responsabilidad del programador (tipea la tecla de al lado, equivocadamente), dado que el programa funcionará pero no se obtendrán los resultados deseados ni mensajes de error. Suelen ser difíciles de detectar, especialmente en programas grandes.

También suele observarse cuando se define una constante; por ejemplo, si tiene que introducir a=0.89 y escribe:

```
>>a=0,89
```

Obtendrá:

```
a =

     0

ans =

    89
```

Dado que el punto decimal es "." y no "," en Matlab. En este caso, la "," se interpreta como un separador de instrucciones.
Incluso cuando se escriben las sentencias a mano (como en los exámenes) hay que poner atención en estos detalles, ya que el funcionamiento del programa puede ser muy diferente.

5) Estructuras de control de flujo incompletas

Es muy común olvidar escribir los *end* para cerrar estructuras de control de flujo. Típicamente es muy fácil de solucionar, solo basta recordar que se requiere un end por cada estructura de control .
Suponga que crea la siguiente función para calcular el factor de fricción de Fanning, donde falta end al final:

```
function f = Fanning (Re)
%function que calcula el factor de fricción de Fanning
if  Re <= 2100
     f= 64/Re;
elseif  Re <= 1e5
     deno = Re.^0.25;
     f = 0.0791./deno
else
     f = 0.0044
```

Cuando se llama a la function se obtendrá:

```
>> ff = Fanning (80)
??? Error: File: C:\MATLAB7\work\Fanning.m Line: 11 Column: 1
This statement is incomplete.
```

6) Nombres incorrectos de funciones

Un error muy observado es que se utilizan nombres incorrectos de funciones. Para la function anterior:

```
function f = Fanning (Re)
%function que calcula el factor de fricción de Fanning
if  Re <= 2100
     f= 64/Re;
elseif  Re <= 1e5
     deno = Re.^0.25;
     f = 0.0791./deno
else
     f = 0.0044
end
```

Es muy común observar que se guarda la función con un nombre diferente, por ej., factor_fanning.m. Cuando se realiza la llamada a la función, mediante Fanning(valor), Matlab contesta indicando que la variable o función no esta definida, puesto que no encuentra el **nombre del archivo**. La práctica recomendada es utilizar el mismo nombre en la línea de definición como para el archivo de la función.

7) Nombre de variables y funciones

En ocasiones el usuario utiliza nombres para sus variables o funciones que ya existen en el entorno MATLAB.

Si se asigna una variable, ones=5, y luego quiere utilizarse la función ones (que genera una matriz de unos), se obtendrá un mensaje de error. Esto puede solucionarse borrando el espacio de trabajo.

Una situación mas comprometida es cuando se genera una función con el mismo nombre que una preexistente.

Para evitar estos inconvenientes, antes de utilizar un *nombre* para una variable o una función, si se cree que puede existir conflicto con funciones existentes, conviene tipear *help nombre* en la ventana de comandos y si existe, no utilizar ese nombre.

8) Argumentos de funciones

Cuando se trabaja con funciones hay que asignar correctamente los argumentos, por ejemplo, considérese nuevamente el factor de fricción:

```
function f = Fanning (Reynolds)
%function que calcula el factor de fricción de Fanning
if  Re <= 2100
     f= 64/Re;
elseif  Re <= 1e5
     deno = Re.^0.25;
     f = 0.0791./deno
else
     f = 0.0044
end
```

Cuando se llama a la function se obtiene:

```
>>Fanning(50)
??? Undefined function or variable 'Re'.
```

```
Error in ==> Fanning at 3
if  Re <= 2100
```

Como el argumento de entrada se llama “Reynolds” y en el cuerpo de la function se utiliza “Re”, esta variable no esta definida.
Lo mismo sucede con los argumentos de salida. Por ejemplo:

```
function f_friccion = Fanning (Re)
%function que calcula el factor de fricción de Fanning
if  Re <= 2100
     f= 64/Re;
elseif  Re <= 1e5
     deno = Re.^0.25;
     f = 0.0791./deno
else
     f = 0.0044
end
```

Cuando se llama a la function se obtiene:

```
>> ff = Fanning(80)
??? One or more output arguments not assigned during call to
'C:\MATLAB7\work\Fanning.m (Fanning)'.
```

Indicando claramente que hay errores en la asignación del argumento de salida (f_fricción debería ser el resultado de la función y no *f)*.

9) Errores en asignaciones

En ocasiones el usuario trata de escribir asignaciones que carecen de sentido, como tratar de asignar valores a un número, o realizar operaciones a la izquierda del signo igual.

Asignar un valor a un número:

```
>> 3=a
??? 3=a
     |
Error: The expression to the left of the equals sign is not a valid
target for an assignment.

>> a+3=b
??? a+3=b
      |
Error: The expression to the left of the equals sign is not a valid
target for an assignment.
```

10) No probar los programas hasta el final

Típicamente cuando el alumno debe resolver un problema, programa toda la secuencia que considera necesaria sin ir “probando” o “verificando” las diferentes “partes” o “módulos” de la misma. Si bien esta práctica no constituye un error, suele inducir la aparición de errores, los cuales son más difíciles de detectar una vez que la secuencia se finalizó. En estas circunstancias el alumno debe corregir gradualmente los errores y muchas veces modificar notablemente el código que inicialmente consideró correcto (lo cual puede producir cierto desconcierto y frustración).

Hasta que el alumno adquiera experiencia, es más conveniente programar por partes e ir verificando el “correcto” funcionamiento de cada sentencia o conjunto de sentencias en particular.

11) No borrar el espacio de trabajo entre pruebas

En numerosas ocasiones se obtienen errores por no borrar el espacio de trabajo. Considere por ejemplo:

```
x=0:.1:10;

for i=1:length(x)
    y(i)=sin(x(i));
end
plot(x,y)
```

Si luego cambia el vector x, de manera que tiene diferente número de elementos:

```
x=0:.5:10;

for i=1:length(x)
    y(i)=sin(x(i));
end
plot(x,y)
```

Se obtendrá:

```
??? Error using ==> plot
Vectors must be the same lengths.

Error in ==> ejemplo at 6
plot(x,y)
```

Esto se produce porque la primera vez que se genera el vector y, el mismo posee 101 elementos, al igual que x, mientras que la segunda vez x posee 21 elementos y los primeros 21 elementos de y son modificados, pero sigue teniendo 101 elementos, por lo cual se produce un mensaje de error.
Esta clase de errores puede desconcertar al alumno, dado que la secuencia de comandos es correcta, pero olvida que ya existen valores para las variables en el espacio de trabajo.

Para evitar este tipo de errores, se puede utilizar “clear all” al inicio del script, por lo cual las variables se borraran siempre antes de comenzar.

```
clear all

x=0:.5:10;

for i=1:length(x)
    y(i)=sin(x(i));
end
plot(x,y)
```

En ocasiones “clear all” puede omitirse, cuando las asignaciones son “vectorizadas”

```
x=0:.5:10;

y=sin(x);
```

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

De esta manera la asignación de “y=” borrará los valores en memoria y no habrá dificultades, en lugar de la asignación “y(i)=”.

Note que este tipo de errores se produce cuando se reduce la longitud o el tamaño de una de las variables. Si primero se utiliza:

```
x=0:.5:10;

for i=1:length(x)
    y(i)=sin(x(i));
end
plot(x,y)
```

y luego

```
x=0:.1:10;

for i=1:length(x)
    y(i)=sin(x(i));
end
plot(x,y)
```

No se obtendrá ningún error, ya que ahora la variable “y” tiene más elementos que la primera vez que se generó.

12) No utilizar la ayuda de funciones

En numerosas ocasiones el alumno solicita ayuda para utilizar funciones, sin antes utilizar los comandos de ayuda de Matlab, que suelen ser muy útiles. Recuerde primero que todo soicitar la ayuda del programa. Por ejemplo

>> help global

13) Directorios de trabajo

En numerosas ocasiones el alumno modifica un archivo y sigue obteniendo el mismo resultado. Esta situación se produce cuando se utiliza más de una versión del mismo archivo y modifican una versión que no es la que está en el directorio de trabajo de MATLAB.

Por ej., puede existir una versión del archivo en el directorio C:\MATLAB6p1\work\ y otra versión en el directorio A:\ o F:\ (dispositivos de almacenamiento portátiles). El alumno debe asegurarse que el archivo que esté modificando se halle en el directorio actual de trabajo de Matlab.

Incluso en ocasiones se producen errores debido a que el alumno llama a una “function” que no está en el directorio de trabajo y no percibe esa situación. Recuerde que el contenido del directorio de trabajo puede verificarse fácilmente con el comando dir.

EPÍLOGO

A lo largo de las páginas de este libro/curso de Matlab, revisadas y actualizadas en función de la experiencia aquilatada en el proceso de enseñanza/aprendizaje, y de las críticas constructivas y observaciones de los alumnos y docentes, se han recorrido los que se consideran temas esenciales para entrenarse en el uso de este entorno de cálculo y visualización gráfica vinculado a un lenguaje de programación de muy alto nivel para el desarrollo de aplicaciones del usuario. Habrá observado que es imprescindible un estudio teórico práctico combinado con resoluciones de problemas de seminario que en ocasiones resultan más provechosas si se resuelven con el entorno Matlab abierto para ir intentando obtener resultados y comprobar que los programas funcionen. Es posible que al principio se haya sentido un poco frustrado por la sucesión de errores que el entorno nos señala de las expresiones que escribimos pero, a medida que se logra la familiarización con el entorno y la programación, aparece cierto automatismo de trabajo que permite concentrarse en resolver el problema técnico, y no dispersarse con las dificultades instrumentales. Dada la posición que tiene la parte de Matlab en la asignatura Simulación de Procesos I SDP1 del Área Departamental Ingeniería Química, Facultad de Ingeniería, UNLP, dentro del 5º semestre, los alumnos recién están comenzando a manejar los conceptos propios de la ingeniería química. Por tanto, los problemas a resolver se han referido con frecuencia a conocimientos de asignaturas previas de ciencias básicas o química general, mezclados con algunas aplicaciones para la carrera. Los temas de la asignatura SDP1 que siguen al Curso de Matlab, constan de Métodos Numéricos aplicados a Ingeniería Química; con ellos, el alumno ganará capacidad para resolver una gran variedad de problemas que no pueden abordarse con métodos analíticos o exactos, y, una vez que se haya acostumbrado a resolverlos con calculadora de mano, podrá implementar los algoritmos en la computadora, utilizando el lenguaje Matlab. De esta manera podrá poner los conocimientos de programación de este curso al servicio de la resolución de problemas prácticos similares a los que podrá encontrar en asignaturas más avanzadas y en la vida profesional, como resolver ecuaciones no lineales y sistemas de ellas, sistemas de ecuaciones lineales, ajuste de curvas lineales y no lineales por cuadrados mínimos, integración numérica o cuadratura, resolución numérica de ecuaciones diferenciales ordinarias y sistemas de ellas, como las usadas para diseñar reactores químicos, y finalmente, ecuaciones diferenciales parciales, que permiten, por ejemplo, calcular tiempos de proceso (y por tanto dimensiones de equipos para muy variadas aplicaciones, desde esterilización de alimentos, a cocción, a deshidratación, reacciones químicas catalíticas, etc.). Es importante que el alumno recuerde que sólo podrá mantener la destreza de resolver problemas mediante programación, si mantiene su práctica en forma razonablemente cotidiana. En este sentido, la programación no es muy distinta al piano o al tenis: sólo se lo puede hacer bien si se lo practica, y se lo sigue practicando. Es importante que los ingenieros programen, puesto que es un medio creativo que permite un espacio de concentración para lograr nuestro fin: aprender nuestra profesión resolviendo problemas concretos. Recuerdo un proverbio chino que leí en un libro de programación: lo grande se reduce a lo pequeño, y lo pequeño se reduce a la nada. Es por eso que hay que mantener la dedicación y la práctica en programación a efectos de posibilitar su utilización para la resolución de problemas, diseño de equipos, y la optimización y control de su operación.

Sergio A. Giner
La Plata, Abril de 2008

REFERENCIAS BIBLIOGRÁFICAS

García de Jalón, J., Rodríguez, J.I., Brazález, A. (2001). Aprenda Matlab 6.1 como si estuviera en primero. ETSII, Universidad Politécnica de Madrid, 113 pp. (www.tayuda.com/ayudainf/aprendainf/varios.htm)

Giner, S.A. (1995). Clases teóricas de Turbo Pascal 6. Cátedra de Programación Avanzada en Ingeniería Química. Departamento de Ingeniería Química, Facultad de Ingeniería, Universidad Nacional de La Plata.

López Román, L. (1994). Programación Estructurada: Un Enfoque Algorítmico. México: Alfaomega, 666 pp.

Moore, H. (2007). Matlab para Ingenieros. Primera Edición. Pearson Educación, México, 624 pp.

Pérez, C. (2002). Matlab y sus Aplicaciones en las Ciencias y la Ingeniería. Pearson Educación, S.A., Madrid, 632 pp.

AGRADECIMIENTOS

Al Ing. Sandro Goñi, Docente de la Cátedra, por su valioso aporte.