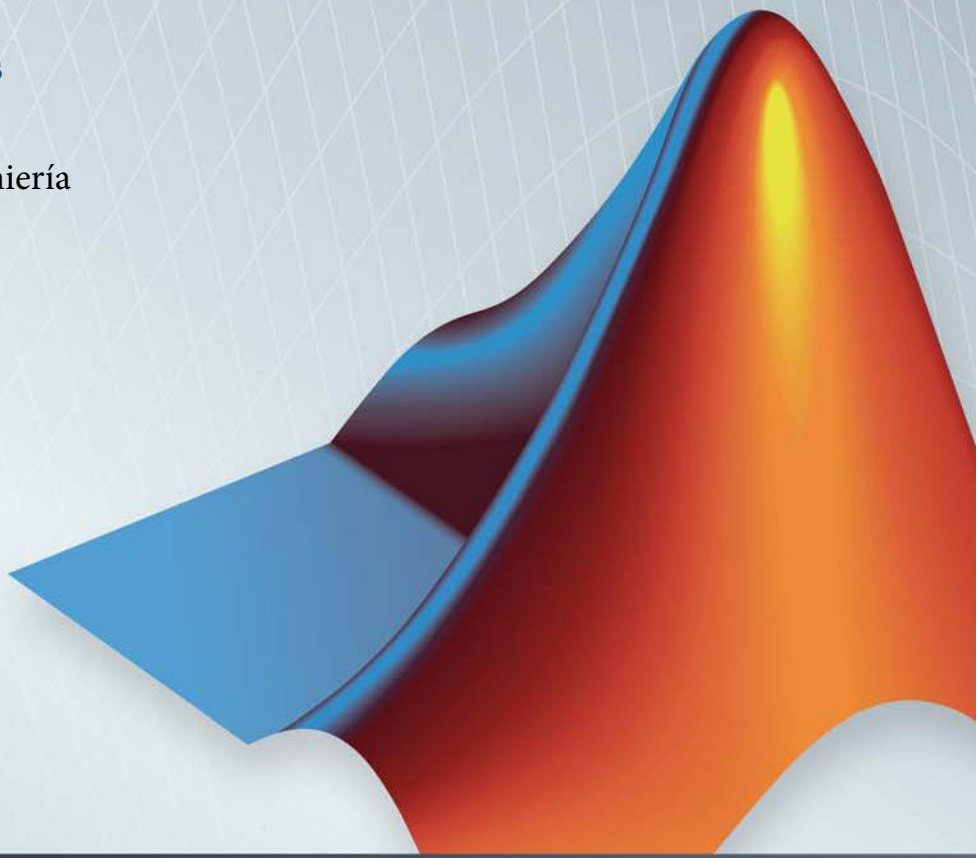
UCH
Universidad de Ciencias y Humanidades

Facultad de Ciencias e Ingeniería



MATLAB® & SIMULINK

PARA INGENIERÍA

NIVEL I

Miguel Ataurima Arellano

mataurimaa@uni.pe

JULIO 2013

MathWorks

Índice general

Capítulo 1

El Entorno de Trabajo de MATLAB

1.1. ¿Qué es MATLAB?

MATLAB es un lenguaje de programación de alto nivel orientado al cálculo técnico que integra un entorno amigable para el cálculo, la visualización de resultados y la codificación de programas.

Generalmente es utilizado en:

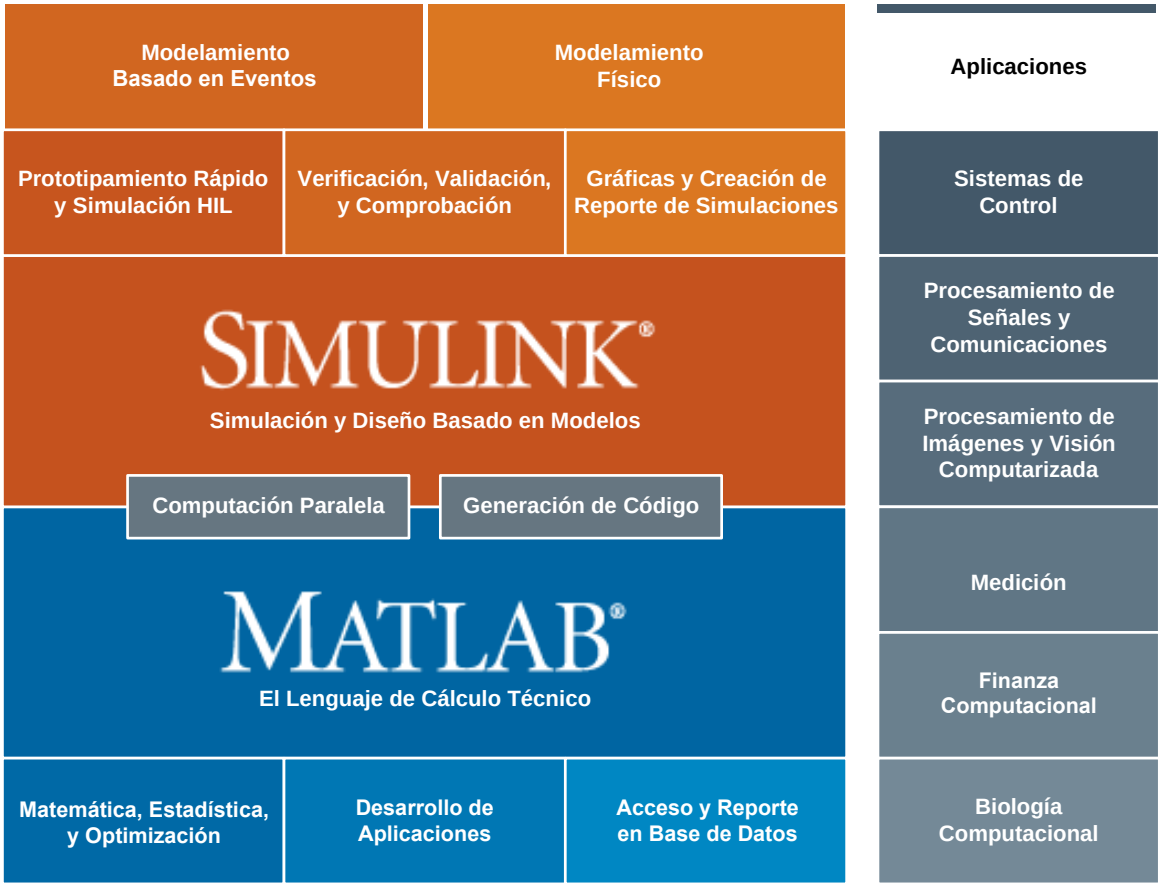
- Cálculo y Matemática
- Desarrollo de Algoritmos
- Adquisición de datos
- Modelamiento, simulación y prototipamiento.
- Análisis, exploración y visualización de datos.
- Gráficos científicos y de ingeniería.
- Desarrollo de aplicaciones con interfaces grácas.

El nombre MATLAB proviene de Matrix Laboratory (Laboratorio de Matrices) dado que en sus orígenes fue escrito para facilitar el desarrollo de software matricial. MATLAB ha evolucionado desde 1970 a través de la atención de las necesidades de sus principales usuarios, tanto en ámbitos académicos como empresariales.

1.1.1. Principales Características

1. Lenguaje de programación de alto nivel para cálculo técnico.
2. Entorno de desarrollo para la gestión de código, archivos y datos.
3. Herramientas interactivas para exploración, diseño y resolución de problemas iterativos.
4. Funciones matemáticas para álgebra lineal, estadística, análisis de Fourier, filtraje, optimización e integración numérica.
5. Funciones gráficas para visualización de datos en 2D y 3D.
6. Herramientas para crear interfaces grácas de usuario personalizadas.
7. Funciones para integrar algoritmos basados en MATLAB con aplicaciones y lenguajes externos (C/C++, FORTRAN, Java, COM y Microsoft Excel).
8. Provee Toolboxes, herramientas orientadas a problemas especícos.

1.2. La Familia de Productos



1.2.1. Productos MATLAB

MATLAB es un lenguaje de programación de alto nivel y un entorno interactivo para el cálculo técnico, e incluye funciones para el desarrollo de algoritmos, análisis de datos, cálculo numérico, y visualización.

MATLAB sirve como base de todos los otros productos de MathWorks.

1. **Matemática, Estadística, y Optimización**

   - **Symbolic Math Toolbox:** Realiza cálculos matemáticos simbólicos.
   - **Partial Differential Equation Toolbox:** Resuelve ecuaciones diferenciales parciales usando métodos de elementos finitos.
   - **Statistics Toolbox:** Realiza modelamiento y análisis estadistico.
   - **Curve Fitting Toolbox:** Ajusta curvas y superficies a los datos usando regresión, interpolación y suavizamiento.
   - **Optimization Toolbox:** Resuelve problemas de optimización estándar y de gran escala.
   - **Global Optimization Toolbox:** Resuelve problemas de optimización de múltiple máximo, múltiple mínimo y sin suavizamiento.
   - **Neural Network Toolbox:** Crea, entrena, y simula redes neuronales.
   - **Model-Based Calibration Toolbox:** Calibra complejos sistemas de propulsión.

2. **Desarrollo de Aplicaciones**

   - **MATLAB Compiler:** Construye ejecutables standalone y componentes de software a partir de código MATLAB.
   - **MATLAB Builder NE:** Desarrolla código MATLAB así como componentes .NET o COM.
   - **MATLAB Builder JA:** Desarrolla código MATLAB así como clases Java.

- **MATLAB Builder EX:** Desarrolla código MATLAB como add-ins de Microsoft Excel.
- **Spreadsheet Link EX:** Permite usar MATLAB desde Microsoft Excel.
- **MATLAB Production Server:** Ejecuta programas MATLAB como una parte de una web, base de datos o aplicaciones corporativas.

3. **Acesso a base de datos y Documentación**

   - **Database Toolbox:** Intercambio de datos con bases de datos relacionales.
   - **MATLAB Report Generator:** Genera documentación para aplicaciones MATLAB y para los datos.

1.2.2. Productos Simulink

Simulink es un entorno de diagramas de bloques extendible para la simulación de sistemas y el diseño basado en modelos. Permite a los ingenieros simular y analizar una amplia gama de sistemas que incluyen controles, señales y procesamiento de imágenes, comunicaciones, y sistemas físicos multidominio.

1. **Modelamiento Basado en Eventos**

   - **Stateflow:** Diseño y simulación de máquinas de estado.
   - **SimEvents:** Modela y simula sistemas de eventos discretos.

2. **Modelamiento Físico**

   - **Simscape:** Modela y simula sistemas físicos multidominio.
   - **SimMechanics:** Modela y simula sistemas mecánicos multibody.
   - **SimDriveline:** Modela y simula sistemas mecánicos unidimensionales.
   - **SimHydraulics:** Modela y simula sistemas hidráulicos.
   - **SimRF:** Diseña y simula sistemas RF.
   - **SimElectronics:** Modela y simula sistemas electrónicos y mecatrónicos.
   - **SimPowerSystems:** Modela y simula sistemas eléctricos de potencia.

3. **Prototipamiento y Simulación HIL rápido**

   - **xPC Target:** Realizar simulación de hardware-in-the-loop en tiempo real de prototipado rápido.
   - **xPC Target Embedded Option:** Ejecuta aplicaciones xPC Target en computadoras destino independientes.
   - **Real-Time Windows Target:** Ejecuta modelos Simulink en tiempo real sobre computadoras con Microsoft Windows.

4. **Verificación, Validación and Prueba**

   - **Simulink Verification and Validation:** Verify models and generated code.
   - **Simulink Design Verifier:** Identify design errors, generate test vectors, and verify designs against requirements.
   - **SystemTest:** Manage tests and analyze results for system verification and validation.
   - **HDL Verifier:** Verify VHDL and Verilog using HDL simulators and FPGA-in-the-loop test benches.
   - **Simulink Code Inspector:** Automate source code reviews for safety standards.
   - **Polyspace Client for C/C++:** Prove the absence of run-time errors in source code.
   - **Polyspace Server for C/C++:** Perform code verification on computer clusters and publish metrics.
   - **Polyspace Client for Ada:** Prove the absence of run-time errors in source code.

- **Polyspace Server for Ada:** Perform code verification on computer clusters and publish metrics.
- **Polyspace Model Link SL:** Trace Polyspace results to Simulink models.
- **Polyspace Model Link TL:** Trace Polyspace results to dSPACE TargetLink blocks.
- **Polyspace UML Link RH:** Trace Polyspace results to IBM Rational Rhapsody models.
- **DO Qualification Kit:** Qualify Simulink and Polyspace verification tools for DO-178 and DO-278.
- **IEC Certification Kit:** Qualify code generation and verification tools for ISO 26262 and IEC 61508 certification

5. **Gráficas y desarrollo de reportes de Simulación**

- **Simulink 3D Animation:** Anima y visualiza modelos en tres dimensiones.
- **Gauges Blockset:** Señales monitores con instrumentos gráficos.
- **Simulink Report Generator:** Genera documentación para Simulink y modelos Stateflow.

1.2.3. Aplicaciones

1. **Sistemas de Control**

- **Control System Toolbox:** Diseña y analiza sistemas de control.
- **System Identification Toolbox:** Crea modelos de sistemas dinámicos lineales y no lineales a partir de datos de medidas de entrada-salida.
- **Fuzzy Logic Toolbox:** Diseña y simula sistemas de lógica difusa.
- **Robust Control Toolbox:** Diseña controladores robustos para plantas inciertas.
- **Model Predictive Control Toolbox:** Diseña y simua modelos de controladores predictivos.
- **Aerospace Toolbox:** Estándares de referencia Aerospacial, modelos de entorno, e importación de coeficientes aerodinámicos.
- **Simulink Control Design:** Ganancias de cálculo PID, modelos linealizados, y diseño de sistemas de control.
- **Simulink Design Optimization:** Estimación y optimización de parámetros de un modelo Simulink.
- **Aerospace Blockset:** Modela y simular aviones, vehículos espaciales y sistemas de propulsión.

2. **Procesamiento de Señales y Comunicaciones**

- **Signal Processing Toolbox:** Lleva a cabo procesamiento de señales, análisis y desarrollo de algoritmos.
- **DSP System Toolbox:** Design and simulate signal processing systems.
- **Communications System Toolbox:** Design and simulate the physical layer of communication systems.
- **Wavelet Toolbox:** Analyze and synthesize signals and images using wavelet techniques.
- **RF Toolbox:** Design, model, and analyze networks of RF components.
- **Phased Array System Toolbox:** Design and simulate phased array signal processing systems.
- **SimRF:** Design and simulate RF systems.
- **Computer Vision System Toolbox:** Design and simulate computer vision and video processing systems

3. **Procesamiento de Imágenes y Visión Computarizada**

- **Image Processing Toolbox:** Perform image processing, analysis, and algorithm development.

- **Computer Vision System Toolbox:** Design and simulate computer vision and video processing systems.
- **Image Acquisition Toolbox:** Acquire images and video from industry-standard hardware.
- **Mapping Toolbox:** Analyze and visualize geographic information

4. **Pruebas y Medición**

- **Data Acquisition Toolbox:** Connect to data acquisition cards, devices, and modules.
- **Instrument Control Toolbox:** Control and communicate with test and measurement instruments.
- **Image Acquisition Toolbox:** Acquire images and video from industry-standard hardware.
- **OPC Toolbox:** Read and write data from OPC servers and data historians.
- **Vehicle Network Toolbox:** Communicate with in-vehicle networks and access ECUs using CAN and XCP protocols

5. **Finanza Computacional**

- **Financial Toolbox:** Analiza datos financieros y desarrollo de modelos en finanzas.
- **Econometrics Toolbox:** Modela y analiza sistemas financieros y económicos usando métodos estadísticos.
- **Datafeed Toolbox:** Accede a datos financieros desde proveedores de servicios de datos.
- **Database Toolbox:** Intercambia datos con bases de datos relacionales.
- **Spreadsheet Link EX:** Usa MATLAB desde Microsoft Excel.
- **Financial Instruments Toolbox:** Diseña, valoriza, y coberturiza instrumentos financieros complejos.
- **Trading Toolbox:** Acceso a precios y envío de órdenes a los sistemas de comercio.

6. **Biología Computacional**

- **Bioinformatics Toolbox:** Read, analyze, and visualize genomic and proteomic data.
- **SimBiology:** Model, simulate, and analyze biological systems.

1.3. Los creadores

- **Cleve Moler.** Director científico y co-fundador de The MathWorks. Es autor de la primera versión de MATLAB y co-autor de las bibliotecas de subrutinas LINPACK y EISPACK (ampliamente utilizadas en todo el mundo). Bachiller en Matemáticas por Caltech (1961), Magister (1963) y Ph.D. (1965) en Matemáticas por la Universidad de Stanford. Ha sido profesor de Matemáticas y Ciencias Computacionales por más de 20 años en universidades las Universidades de Michigan, Stanford y Nuevo Méjico. Trabajó para Intel Hypercube y Ardent Computer Corporation. Es coautor de varios textos sobre métodos numéricos y miembro de la ACM.

- **Jack Little.** Presidente y co-fundador de The MathWorks. Coautor y principal arquitecto de las versiones inciales de MATLAB, Signal Processing Toolbox y Control Systems Toolbox. Bachiller en Ingeniería Eléctrica por el MIT (1978) y Magister por la Universidad de Stanford (1980). Es miembro de la IEEE. Se encarga de la escritura y divulgación de los calculos técnicos, diseños nasado en modelos, y temas de la industria del software.

1.4. Las versiones

Año	Versión	Nombre liberado
1984	MATLAB 1.0	
1986	MATLAB 2	
1987	MATLAB 3	
1990	MATLAB 3.5	
1992	MATLAB 4	
1994	MATLAB 4.2c	R7
1996	MATLAB 5.0	R8
1997	MATLAB 5.1	R9
	MATLAB 5.1.1	R9.1
1998	MATLAB 5.2	R10
	MATLAB 5.2	R10.1
1999	MATLAB 5.3	R11
	MATLAB 5.3.1	R11.1

Año	Versión	Nombre liberado
2000	MATLAB 6.0	R12
2001	MATLAB 6.1	R12.1
2002	MATLAB 6.5	R13
2003	MATLAB 6.5.1	R13SP1
	MATLAB 6.5.2	R13SP2
2004	MATLAB 7	R14
	MATLAB 7.0.1	R14SP1
2005	MATLAB 7.0.4	R14SP2
	MATLAB 7.1	R14SP3
2006	MATLAB 7.2	R2006a
	MATLAB 7.3	R2006b
2007	MATLAB 7.4	R2007a
	MATLAB 7.5	R2007b

Año	Versión	Nombre liberado
2008	MATLAB 7.6	R2008a
	MATLAB 7.7	R2008b
2009	MATLAB 7.8	R2009a
	MATLAB 7.9	R2009b
2010	MATLAB 7.10	R2010a
	MATLAB 7.11	R2010b
2011	MATLAB 7.12	R2011a
	MATLAB 7.13	R2011b
2012	MATLAB 7.14	R2012a
	MATLAB 8.0	R2012b
2013	MATLAB 8.1	R2013a

1.5. El Sistema MATLAB

1. **Herramientas de Escritorio y Entornos de Desarrollo**

   Ayudan a utilizar con mayor productividad los archivos y funciones MATLAB (el escritorio MATLAB, el editor/depurador, el analizador de código, los navegadores para la visualización de ayuda, el workspace, etc).

2. **La Biblioteca de Funciones Matemáticas**

   Colección de algoritmos computacionales que abarca desde funciones matemáticas básicas hasta las mas sosticadas.

3. **El Lenguaje**

   MATLAB es un lenguaje de alto nivel basado en matrices/arreglos que posee sentencias de control de ujo, funciones, estructuras de datos, entrada/salida, y características de programación orientada a objetos.

4. **Los Gráficos**

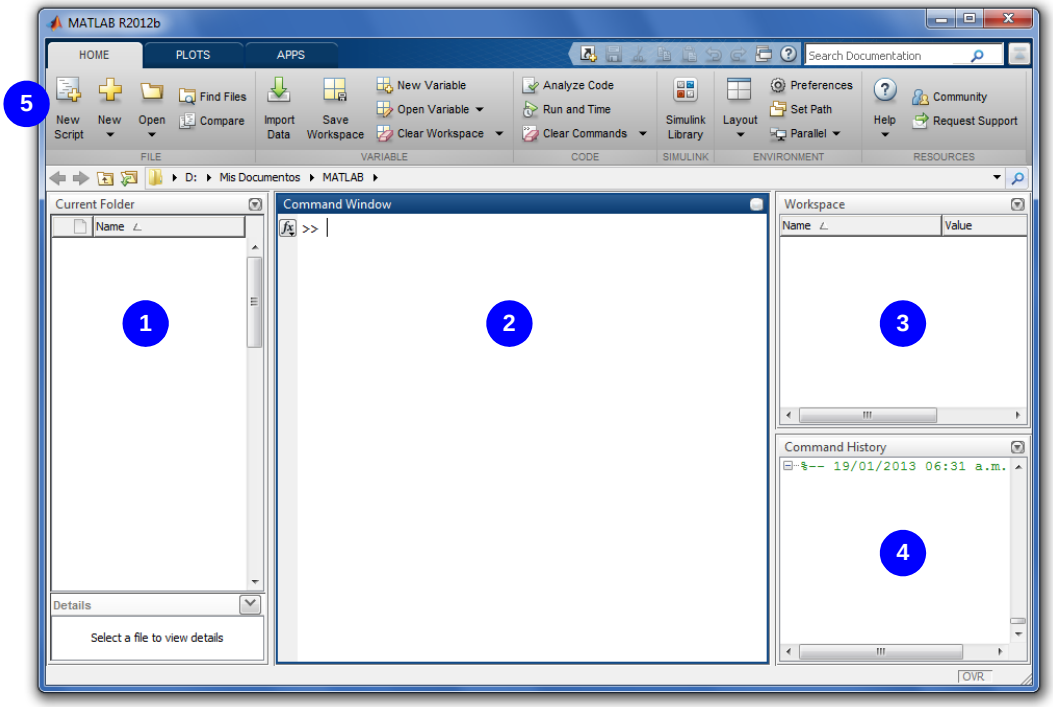
   MATLAB posee funciones de alto y bajo nivel para la visualización de datos en dos y tres dimensiones, así como para el desarrollo de interfaces grácas.

5. **Las Interfaces Externas**

   Las bibliotecas de interfaces externas permiten escribir programas en C y Fortran que interactúen con MATLAB.

1.6. El escritorio MATLAB

Cuando se inicia MATLAB por primera vez, el escritorio (desktop) aparecerá con sus paneles en la disposición (layout) por defecto (default)



1. Current Folder
2. Command Window
3. Workspace
4. Command History
5. Interfaz Ribbon

El Current Folder nos permitirá acceder accede a los archivos. El Command Window es la ventana por medio de la cual se ingresarán líneas de comandos, en el punto de inserción denominado prompt (>>). A traves del Workspace podremos explorar los datos que se vayan creando o importando a partir de archivos. En el Command History visualizaremos o reejecutaremos los comandos que han sido ingresados previamente digitados en la línea de comandos.

1.7. La Ventana de Comandos (Command Window)

La Ventana de Comandos permite el ingreso de datos, la ejecución de código MATLAB y la visualización de resultados. El prompt de la Ventana de Comandos indica el punto en el que podemos darle una entrada a MATLAB. Como el prompt es el punto de inserción de sentencias, éste es también conocido como la línea de comando.

La apariencia del prompt puede tomar diferentes aspectos:

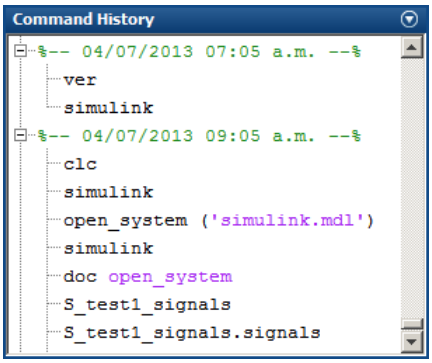
- >> indica que la Ventana de Comando está en modo normal
- EDU>> indica que la Ventana de Comando está en modo normal, en la Versión de Estudiante de MATLAB
- K>> indica que MATLAB está en modo de depuración (debug mode)

1.8. La Ventana Historial de Comandos (Command History)

La Ventana Historial de Comandos visualiza un registro (log) de sentencias que se hayan ejecutado en las sesiones MATLAB actual y previas. La hora y la fecha de cada sesión aparece al inicio de las sentencias listadas para aquella sesión. Todas las entradas son registadas en el archivo history.m.

El archivo history.m:

- Reside en la carpeta que nos retorna el comando prefdir
- Se carga cuando MATLAB inicia.
- Almacena un máximo de 200,000 bytes
- Elimina las entradas mas antiguas necesarias tal que se mantenga el número máximo de bytes.



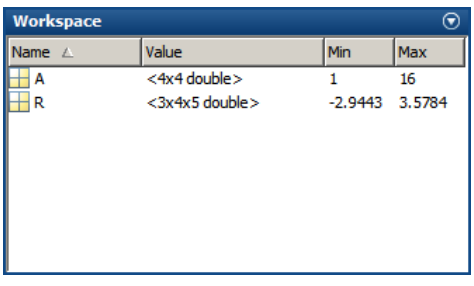
1.9. El Workspace Browser

El Workspace de MATLAB (MATLAB Workspace) es el conjunto de variables creadas y almacenadas en memoria durante una sesión MATLAB. Cuando se utilizan funciones, ejecuta código MATLAB y se carga workspaces almacenados, se añaden variables al workspace. El Workspace Browser es una herramienta que nos permite ver, modicar, y gracar valores del Worskpace de MATLAB. Por defecto, el Workspace Browser muestra el Worskpace base. Si MATLAB se halla en modo de depuración (debug mode), el campo Stack nos permitirá ver los worskpaces de las funciones.

Por ejemplo, si se ejecutan las sentencias:

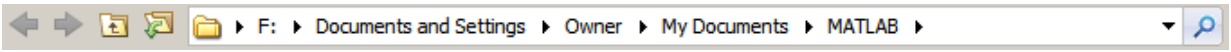
```
A = magic(4)
R = randn(3,4,5)
```

el workspace añadira dos variables, A y R.



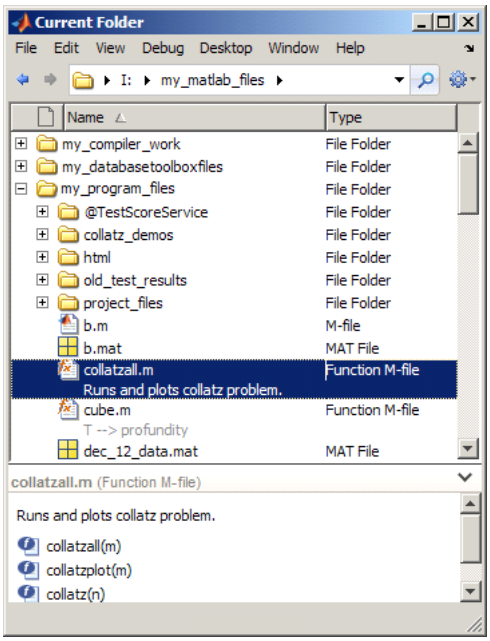
1.10. La Ventana Carpeta Actual (Current Folder)

La Carpeta de Inicio (Startup folder) es la Carpeta Actual (Current folder) en la aplicación MATLAB cuando ésta inicia.



La Carpeta Actual debe ser la Carpeta que se usa con mayor frecuencia.
La Ventana Carpeta Actual (Current Folder):

- Muestra siempre la ruta completa de la carpeta actual en la barra de dirección, y el contenido de la actual carpeta en un así como sus respectivas subcarpetas en el panel debajo de la barra.
- Permite el acceso a las características de gestión de archivos del sistema operativo desde dentro de MATLAB.
- Es similar a los administradores de archivos provistos por los sistemas operativos, añadiendo características únicas para MATLAB.



1.11. Los Atajos de Teclado (Keyboard shortcuts)

MATLAB provee atajos de teclados para navegar en un historial de comandos y listar ayudas contextuales.

1. **La Tecla Flecha Arriba**

   Suponga que por error se digitó la sentencia

   ```
   >> y=sine(pi/4)
   ```

   MATLAB retornará:

   ```
   ??? Undened function or method sinefor input arguments of type double.
   ```

   noticando que error se dá al invocar una inexistente función o método sine.

   Entonces, para no tener que retipear la sentencia evitando cometer el error (digitar sin en vez de sine), bastará con presionar la tecla echa arriba, y la línea que conteniene el error será mostrada. Luego, usando la tecla echa izquierda, movemos el cursor, corregimos el error y presionamos ENTER para ejecutar el comando corregido (nuevo comando).

   ```
   >> y=sin(pi/4)
   y=
       0.7071
   ```

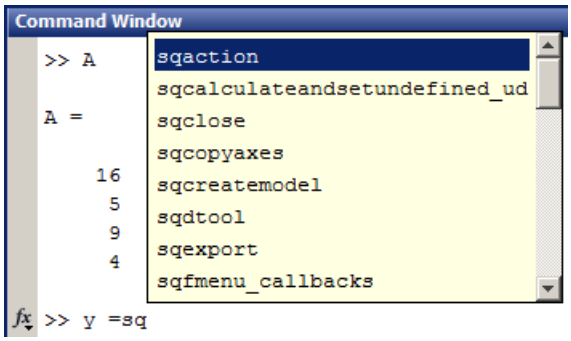
2. **La Tecla Tab**

   Suponga que desea obtener la raíz cuadrada de un número e inicia a digitar la sentencia

   ```
   >> y=sq
   ```

   cuando de pronto se da cuenta que ha olvidado el nombre correcto de la función MATLAB que obtiene la raíz cuadrada !!

   En este punto, bastará con presionar la tecla tab y MATLAB nos presentará un menu contextual conteniendo todas las sentencias que inician con sq.



   Elegimos la sentencia adecuada, indicamos el argumento de la función y nalmente presionando ENTER.

   ```
   >> y=sqrt(2)
   y=
       1.4142
   ```

3. **El operador punto y coma (Semicolon)**

   El operador punto y coma al nal de una linea suprime la salida (eco) en pantalla de MATLAB, o sea la ejecuta en silencio. Esto es útil cuando se desea mantenar limpia la Ventana de Comandos.

   Por ejemplo, al digitar la siguiente entrada y luego presionar ENTER

   ```
   >> x=2+2
   ```

   el resultado de la operación (salida) será mostrado:

   ```
   x=4
   ```

   ahora, reinvocamos nuestra entrada inicial presionando la tecla Flecha Arriba

   ```
   >> x=2+2
   ```

   luego, procedemos a insertar un punto y coma y presionamos ENTER

   ```
   >> x=2+2;
   ```

   Observe que si bien MATLAB no muestra ningún resultado numérico; la sentencia fué ejecutada y se asignó el resultado de la operación 2+2 a la variable x, pudiendo en adelante reinvocar el valor de x, digitando x y presionando ENTER.

1.12. El Sistema de Ayuda de MATLAB

MATLAB tiene tres formas de ayuda en linea: help, doc, y demos.

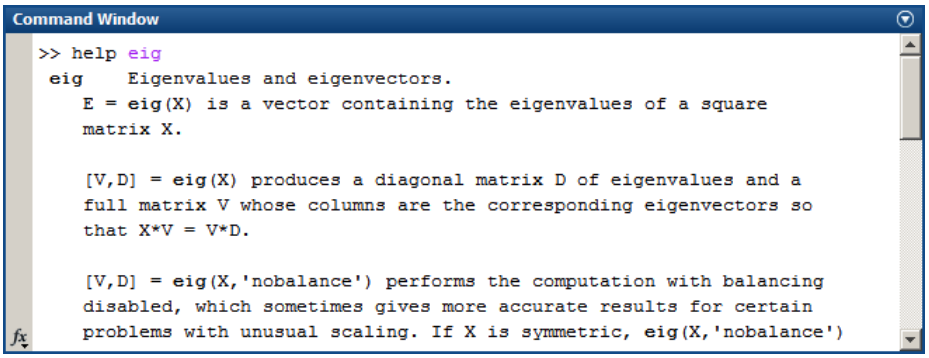
1.12.1. Help

Digitando help en la Ventana de Comandos se visualizará una listado de los tópicos de ayuda mas importantes.

```
>> help
```

Si se desea consultar especícamente por un comando o función, se digita help seguido del comando o función. El resultado obtenido será en formato de texto simple

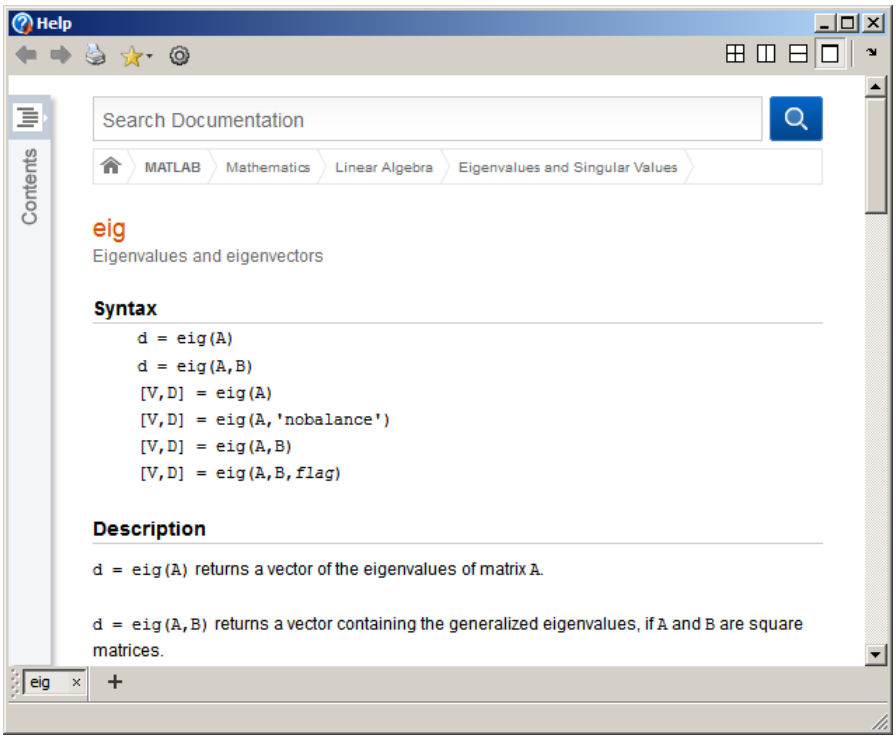
Por ejemplo, para saber acerca del comando eig, digitamos help eig en el la línea de comandos:



1.12.2. Doc

Funciona de manera similar a help, solo que el resultado de las consultas será visualizado en formato HTML en el Help Browser

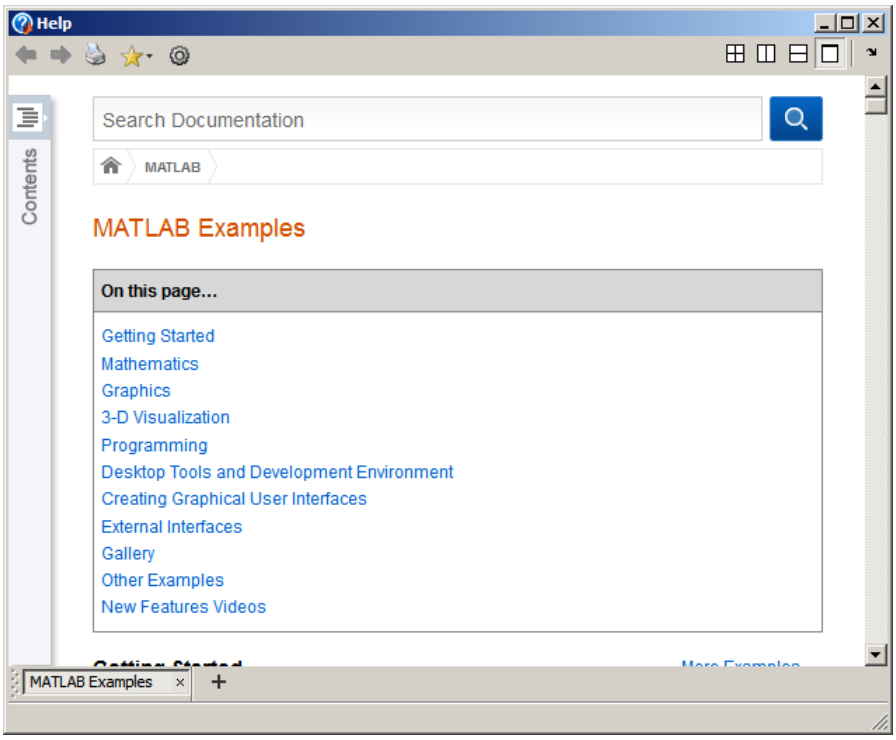
```
>> doc eig
```



1.12.3. Demos

Es una herramienta de ayuda que nos permite aprender mas sobre MATLAB a través de demostraciones. Esto lo hacemos tipeando demo en la línea de comandos, obteniendo un listado de enlaces a demos en el Help Browser.

```
>> demo
```



1.13. Funciones y Comandos útiles

Para una detallada explicación y obtención de ejemplos para cada uno de los siguientes comando/-función consulte la ayuda de MATLAB.

Comando/Función	Significado
`clc`	Limpia la Ventana de Comando
`clear`	Limpia los elementos del Workspace
`who, whos`	Lista las variables del Workspace
`workspace`	Muestra el Workspace browser
`cd`	Cambia del directorio de trabajo (carpeta actual)
`pwd`	Muestra la carpeta actual
`computer`	Retorna información acerca de la computadora en donde MATLAB se está ejecutando
`ver`	Muesta información de la versión de los productos MathWorks instalados
`exit, quit`	Termina la sesión MATLAB

1.14. Principales herramientas del Toolstrip (Cinta de Herramientas)

El Toolstrip organiza la funcionalidad de MATLAB en una serie de pestañas (tabs, en inglés). Las pestanañas están divididas en secciones que contienen una serie de controles relacionados. Los controles son botones, menus desplegables y otros elementos de interfaz de usuario que se utilizan para realizar tareas en MATLAB.

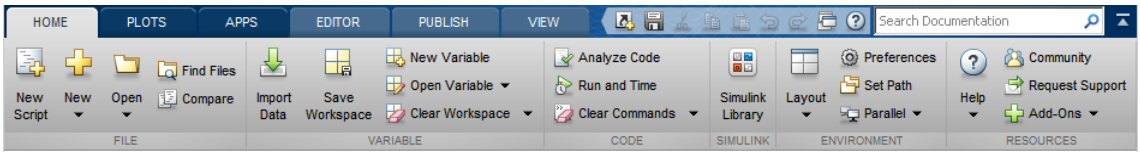
Cuando se inicia MATLAB por primera vez, se observarán tres pestañas: la pestaña Home, la pestaña Plots, y la pestaña Apps. Estas tres pestaña estarán siempre ahí sin importar lo que se esté realizando

en MATLAB. Por esta razon, éstas son llamadas **pestañas globales**.

1.14.1. Las Pestañas Globales

1. **La petaña HOME**

   La pestaña Home es donde ser realizan operaciones de propósito general tales como crear nuevos archivos, importar datos, gestionar el workspace y configurar el diseño del escritorio.

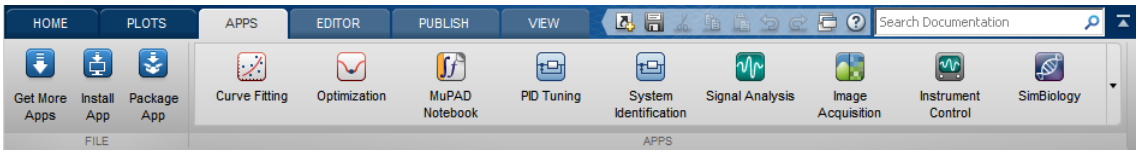


2. **La pestaña PLOTS**

   La pestaña Plots muestra una galería de trazas disponibles en MATLAB y algunos de los toolboxes que se tengan instalados. Para crear una trasa a partir de la galería, se debe seleccionar las variables en el workspace que se desee trazar y luego selecconar el tipo de visualización que se desee para aquellos datos. La flecha que apunta hacia abajo despliega la galería de trazas con muchas mas opciones. La galeria es inteligente, solo muestra las trazas que son apropiadas para los datos seleccionados.

3. **La pestaña APPS**

   La pestaña Apps es el lugar en donde se ejecutan las aplicaciones interactivas MATLAB. Muchas de estas aplicaciones provienen de MathWorks, se obtienen automáticamente con los Toolboxes que se hayan instalado. La pestaña Apps presenta una galería de apps que se haya instalado. La flecha que apunta hacia abajo despliega la galeria de apps con muchas mas opciones. Simplemente, se da clic en la app favorita (por ejemplo, Curve Fitting) y la app inicia. La pestaña Apps reemplaza al viejo menu "Start".

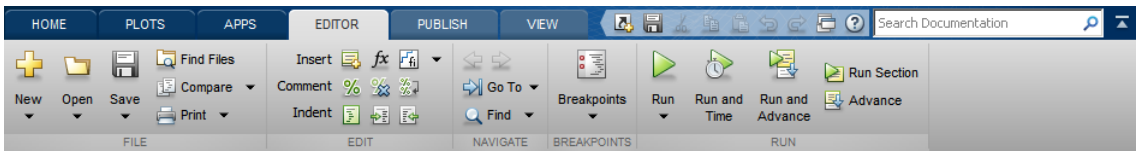


1.14.2. Las Pestañas Contextuales

Las Pestañas Contextuales solo aparecen cuando se realizan ciertas tareas en MATLAB. Por ejemplo, cuando se utiliza el Editor para editar un archivo, aparecen tres nuevas pestañas: la pestaña Editor, la pestaña Publish, y la pestaña View. Si el Editor esta acoplado (docked) en el Escritorio, las pestañas relacionadas con el Editor aparecerán junto a las pestañas globales.

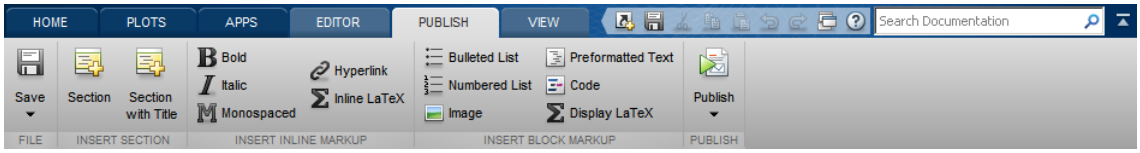
1. **La pestaña EDITOR**

   La pestaña Editor contiene todas las funciones necesarias para editar un archivo. Todas la capacidades del Editor estan organizadas de tal manera que sean fáciles de encontrar y usar.



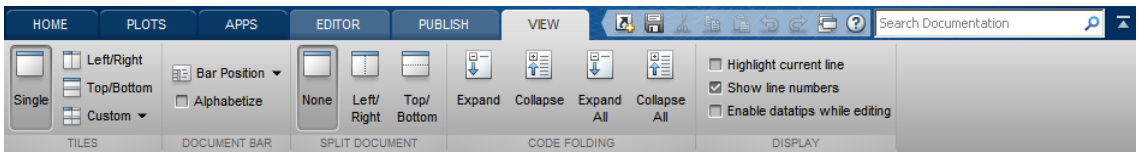
2. **La pestaña PUBLISH**

   La pestaña Publish es otra pestaña asociada al Editor. La pestaña Publish toma todos los controles de formato necesarios para crear documentos MATLAB. La publicación es una característica muy útil en MATLAB que ha estado por muchos años. A pesar de ello, los usuarios de MATLAB no siempre la encuentran.



3. **La pestaña VIEW**

   La pestaña View es el último de las pestañas contextuales. Es a partir de donde se controlan el diseño y la apariencia de los archivos en el Editor. También encontrará pestañas contextuales en el Editor de Variables.



1.14.3. Minimización del toolstrip

Muchas veces se necesita maximizar el espacio vertical de trabajo, siendo útil poder minimizar el Toolstrip. Para ello basta con dar clic derecho en cualquier parte del Toolstrip y seleccionar "Minimize Toolstrip" o dar doble clic en cualquier de las pestañas. Cuando el toolstrip es minimizado éste lucirá así:

Capítulo 2

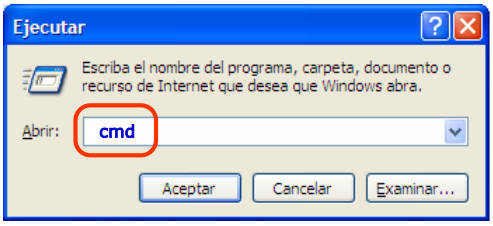
Elementos Básicos del Lenguaje MATLAB

2.1. Los Comandos y las Funciones MATLAB

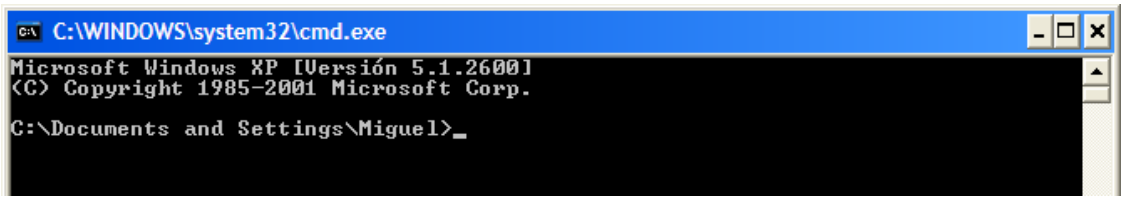
Un comando es una orden o instrucción que el usuario proporciona a un sistema informático, desde la línea de comandos (shell) o desde una llamada de programación.

EJEMPLO: En el Sistema Operativo Windows

- Iniciamos el intérprete de comandos (consola o shell).
  - Damos clic en Inicio y elegimos la opción Ejecutar: Inicio | Ejecutar o sino mediante Inicio | Programas | Accesorios | Símbolo del Sistema
  - Inmediatamente se nos mostrará la ventana Ejecutar



  - En el cuadro de edición Abrir, digitamos cmd; posteriormente, damos clic en Aceptar. Inmediatamente se nos mostrará el Interprete de Comandos de Windows



- La Ventana Intérprete de Comandos de Windows describe el nombre y versión del Sistema Operativo junto con sus derechos de autor; seguido de la línea de comandos (línea de órdenes), desde la cual se ingresan los comandos.
- El conjunto de caracteres que se muestran en la línea de comandos para indicar que el Sistema Operativo está a la espera de órdenes se denomina prompt.

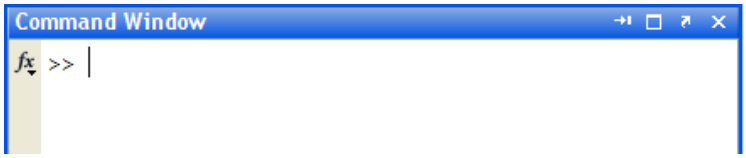
  `C:\Document and Settings\Miguel>`
- El punto de inserción de comandos en el prompt lo establece el cursor representado mediante una barra horizontal (subrayado) intermitente que se halla inmediatamente después del prompt.

  `C:\Document and Settings\Miguel> _`

- Los comandos se digitan desde la posición del cursor; y para su ejecución, éstos deben finalizarse presionando la tecla ENTER.
- Pruebe los siguientes comandos: ver, cls, dir y help.

EJEMPLO: En el Sistema MATLAB

- Elegimos la Ventana de Comandos (intérprete de comandos de MATLAB).



- A diferencia del Intérprete de Comandos de Windows, el directorio actual no se incluye en el prompt; éste es indicado en la barra de herramientas integrando un conjunto de directorios alternativos en la lista desplegable Current Directory.
- El cursor está representado por una barra vertical intermitente.
- Pruebe los siguientes comandos: ver, clc, dir, matlabroot, pwd y help.

NOTAS:

- Durante el procesamiento de un comando; si éste involucra la ejecución de una gran cantidad de instrucciones, se visualizará la palabra Busy a la derecha del botón Start.
- El tiempo de ejecución del comando dependerá de la complejidad de éste, del número del procesos que a la vez éste ejecutando el Sistema Operativo; así como del hardware con que se cuente (la capacidad de memoria, tipo de procesador, etc.)

2.1.1. Los Comandos MATLAB

Los Comandos MATLAB permiten calcular el resultado de una expresión ubicada a la derecha del signo igual, asignando el valor resultante a la variable ubicada a la izquierda (variable de salida).

```
>> y = 4.32*log10(1+0.135)-5
y =
   -4.7624
>>
```

Los comandos MATLAB no mostrarán el valor del resultado asignado a la variable de salida cuando culminen con punto y coma.

```
>> y = 4.32*log10(1+0.135)-5;
>>
```

Si no se asigna explícitamente la salida de un comando a una variable. MATLAB asigna el resultado a la palabra reservada ans.

```
>> 4.32*log10(1+0.135)-5
ans =
   -4.7624
```

El valor de ans varía con cada comando que reporte un valor de salida que no se asigne a variable alguna.

```
>> 4.32*log10(1+0.135)-5
ans =
   -4.7624

>> 3.13^2-sqrt(1/0.4217)
ans =
    8.2570
```

Se puede ingresar mas de un comando en una línea finalizándola con coma (,) o punto y coma (;). Los comandos terminados con coma muestran sus resultados cuando son ejecutados; mientras que los terminados con punto y coma, no.

```
>> d=4/3.14; 1.3^4, exp(-0.31), w=d+ans
ans =
    2.8561
ans =
    0.7334
w =
    2.0073
```

Cuando un comando genera mas de una salida, se debe especificar el conjunto de variables de salida separadas por espacio en blanco o con comas y encerrado, en conjunto, entre corchetes.

EJEMPLO: El comando deal distribuye los valores de cada uno de sus entradas en variables de salida separadas

```
>> [A,B,C] = deal( [-12.3 4.89 -3.01] , pi*1.46, diag(12:4:24) )
A =
  -12.3000 4.8900 -3.0100

B =
   4.5867

C =
   12  0  0  0
    0 16  0  0
    0  0 20  0
    0  0  0 24
```

2.1.2. Las Funciones MATLAB

Ejecutan un conjunto de instrucciones que toman como datos un conjunto de argumentos de entrada y devuelven como resultado un conjunto de argumentos de salida

La sintaxis de una función MATLAB es

```
[v1,v2,...,vm] = nombreFcn(u1,u2,...,un)
```

donde:

- u1,u2,...,un : son los argumentos de entrada de la función
- v1,v2,...,vm : son los argumentos de salida de la función
- nombreFcn : es el nombre de la función

EJEMPLO: Generación de una matriz cuadrada de tamaño 3x3 de elementos aleatorios comprendidos entre 0 y 1.

```
>> A = rand(3)
A =
   0.0971 0.3171 0.4387
   0.8235 0.9502 0.3816
   0.6948 0.0344 0.7655
```

EJEMPLO: Obtención de los vectores propios y valores propios de la matriz del ejemplo anterior.

```
>> [V,D] = eig(A)
V =
   0.7903 -0.3303 -0.0146
  -0.3635 -0.8614 -0.8198
  -0.4932 -0.3860  0.5725

D =
  -0.3225       0       0
        0  1.4369       0
        0       0  0.6985
```

2.2. Los arreglos

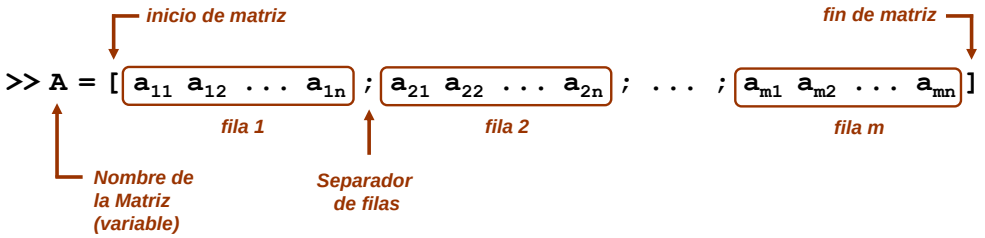
Son elementos dispuestos en dimensiones (en el caso de dimensión 2, se tienen filas y columnas). Estos elementos no necesariamente obedecen a algún conjunto de reglas algebráicas (por ejemplo, las del álgebra lineal), son solo elementos que contienen valores.

Sea el caso mas usual, de dimensión 2, en el que los elementos están dispuestos en forma rectangular con m filas y n columnas

$$A = \underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_{m\times n}$$

fila 1 · fila 2 · fila m

para ingresarla en MATLAB digitamos:



Observaciones:

- Los elementos de una misma fila deben separarse con espacio en blanco o coma (,)
- Todas las filas deben contener n elementos.
- Todos los elementos deben ser de la misma clase

EJEMPLO: Ingresar los siguientes arreglos

$$\mathbf{M} = \begin{bmatrix} -3 & 2 & 1 \\ 0 & 4 & 9 \\ 1 & 7 & 8 \\ 9 & 11 & -3 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} -5 \\ 4 \\ 0 \\ 7 \end{bmatrix}$$

```
>> M = [-3 2 1; 0 4 9; 1 7 8; 9 11 -3]
M =
    -3     2     1
     0     4     9
```

```
     1     7     8
     9    11    -3

>> V = [-5; 4; 0; 7]
V =
    -5      4      0      7
```

Otra forma de haber ingresado el arreglo **V**es como un vector fila al cual se le aplica la transpuesta

```
>> V = [-5 4 0 7] .'
V =
    -5
     4
     0
     7
```

EJEMPLO: Ingresar los siguientes arreglos

$$\mathbf{B} = \begin{bmatrix} 4+i & 2+i \\ -3i & 3-i \end{bmatrix}$$

donde $i = \sqrt{-1}$

```
>> B = [4+i 2+i; -3*i 3-i]
B =
   4.0000 + 1.0000i   2.0000 + 1.0000i
   0.0000 - 3.0000i   3.0000 - 1.0000i
```

En MATLAB las variables i y j están predefinidas con el valor de $\sqrt{-1}$, por lo tanto otra forma de haber ingresado la matriz **B** es

```
>> B = [4+j 2+j; -3*j 3-j]
B =
   4.0000 + 1.0000i   2.0000 + 1.0000i
   0.0000 - 3.0000i   3.0000 - 1.0000i
```

Por otro lado, cuando se usa la unidad imaginaria, la premultiplicación de cualquier número por la variable$i$ o j no requiere de la presencia explícita del operador de multiplicación * (éste es el único caso, en los demás, siempre que se especifique una multiplicación deberá de hacerse de forma explícita)

```
>> B = [4+i 2+j; -3j 3-i]
B =
   4.0000 + 1.0000i   2.0000 + 1.0000i
   0.0000 - 3.0000i   3.0000 - 1.0000i
```

Observación: Basta que uno de los elementos del arreglo sea complejo y Matlab representará a todos los demás también como complejo. Si son reales, tendrán un 0 en la parte imaginaria.

EJEMPLO: Concatenar las matrices

$$\mathbf{M} = \begin{bmatrix} -3 & 2 & 1 \\ 0 & 4 & 9 \\ 1 & 7 & 8 \\ 9 & 11 & -3 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} -5 \\ 4 \\ 0 \\ 7 \end{bmatrix} \quad \mathbf{Q} = \begin{bmatrix} -2 & 1 & 12 & 17 \end{bmatrix}$$

de manera que se obtenga la matriz **R**

$$R = \left[\begin{array}{c|c} \mathbf{M} & \mathbf{V} \\ \hline \multicolumn{2}{c}{\mathbf{Q}} \end{array}\right]$$

```
>> M = [-3 2 1; 0 4 9; 1 7 8; 9 11 -3];
>> V = [-5; 4; 0; 7];
>> Q = [-2 1 12 17];
>> R = [ M V ; Q ]
R =
    -3     2     1    -5
     0     4     9     4
     1     7     8     0
     9    11    -3     7
    -2     1    12    17
```

2.3. Las variables

Una variable está formada por un espacio en el sistema de almacenaje o memoria principal de la computadora, que en MATLAB recibe el nombre de ***workspace***, y un nombre simbólico (un identificador) que está asociado a dicho espacio. Ese espacio contiene una cantidad o información conocida o desconocida, es decir un valor. El nombre de la variable es la forma usual de referirse al valor almacenado: esta separación entre nombre y contenido permite que el nombre sea usado independientemente de la información exacta que representa. El identificador, en el codigo fuente de la computadora puede estar ligado a un valor durante el tiempo de ejecución y el valor de la variable puede por lo tanto cambiar durante el curso de la ejecución del programa.

El concepto de variables en programación puede no corresponder directamente al concepto de variables en matemática. El valor de una variable en programación no es necesariamente parte de una ecuación o fórmula como en matemáticas. En programación una variable puede ser utilizada en un proceso repetitivo: puede asignársele un valor en un sitio, ser luego utilizada en otro, más adelante reasignarsele un nuevo valor para más tarde utilizarla de la misma manera. Procedimientos de este tipo son conocidos con el nombre de iteración.

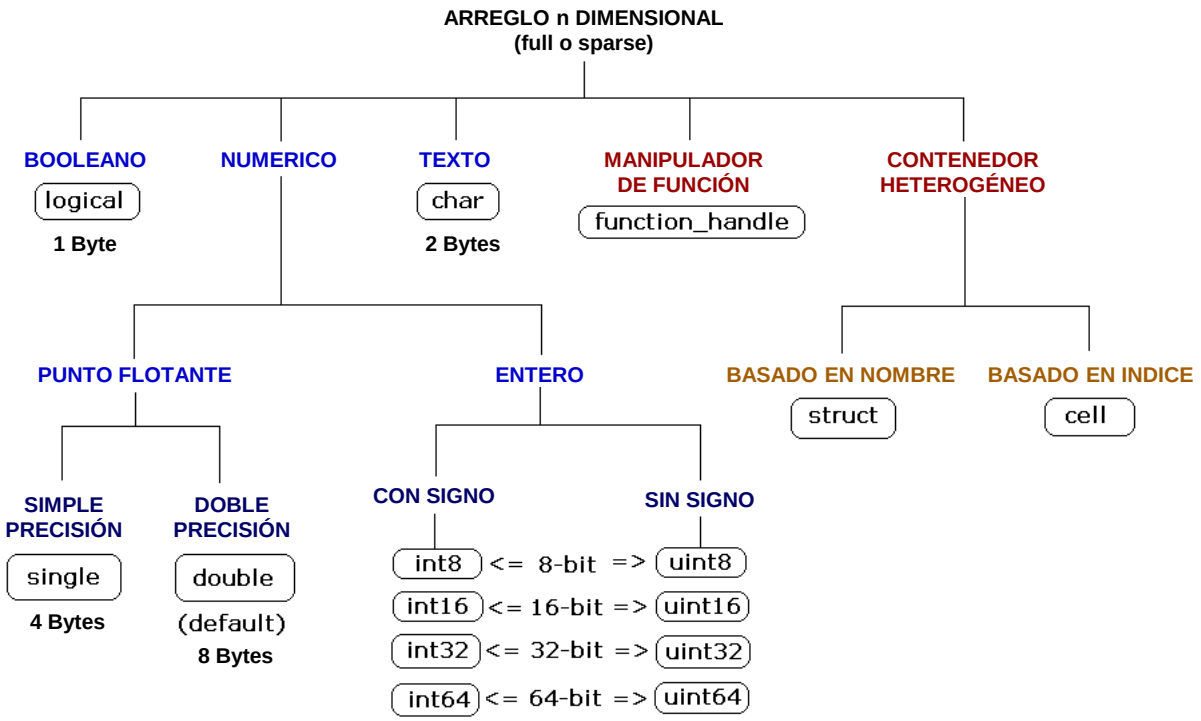
En programación de computadoras, a las variables, frecuentemente se le asignan nombres largos para hacerlos relativamente descriptivas para su uso, mientras que las variables en matemáticas a menudo tienen nombres escuetos, formados por uno o dos caracteres para hacer breve en su transcripción y manipulación.

En MATLAB, las variables pueden ser creadas por código fuente a partir de operaciones que hacen uso de constantes, otras variables o funciones; así como de manera explícita a través de la Ventana de Comandos (tal como se ha visto en los ejemplos anteriores).

2.4. Los tipos de dato (clases)

Tipo de dato informático es un atributo de una parte de los datos que indica al ordenador (y/o al programador) algo sobre la clase de datos sobre los que se va a procesar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

En MATLAB existen 15 tipos de datos (clases) fundamentales. Cada uno de ellos esta dado en forma de un arreglo. Un arreglo puede crecer de tamaño desde 0x0 (Matriz Nula, dimensión 2) hasta otro de cualquier tamaño y de cualquier dimensión.



2.4.1. Combinación de distintos tipos de dato (clases)

Cuando una matriz es compuesta con elementos de distinto tipo de dato, MATLAB convierte algunos elementos de tal manera que todos los elementos de la matriz sean del mismo tipo. La conversión del tipo de dato es efectuada con respecto a la precedencia predefinida de los tipos de datos. La concatenación con distintos tipos de dato sin generación de error se pueden dar solo con cinco de ellos.

TIPO		char	NUMERICO			logical
			entero	single	double	
char		char	char	char	char	**inválido**
	entero	char	entero	entero	entero	entero
	single	char	entero	single	single	single
	double	char	entero	single	double	double
logical		**inválido**	entero	single	double	logical

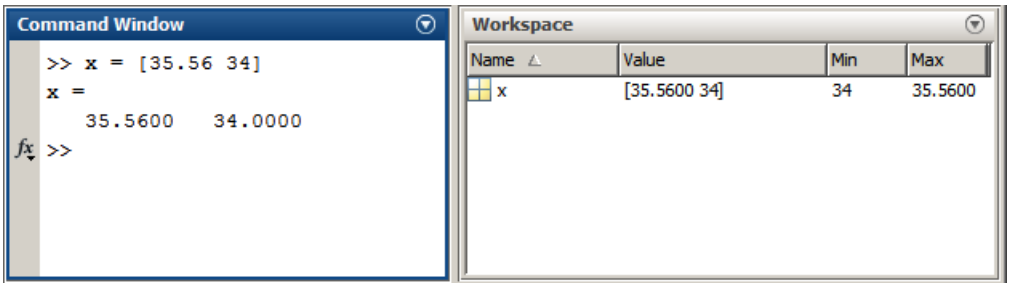
2.5. El workspace

El ***workspace*** es el área de la memoria del sistema donde MATLAB registra a todas las variables que van siendo creadas durante una sesión.

EJEMPLO: Crear la variable

$$\mathbf{x}=\begin{bmatrix} 35{,}56 & 34 \end{bmatrix}$$

y constatar que ésta ha sido almacenada en el workspace.

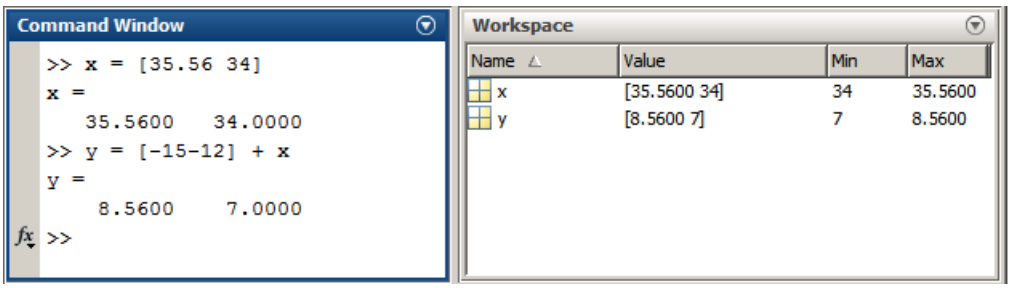


Mientras que una variable esté en el workspace, ésta podrá ser utilizada en otros cálculos.

EJEMPLO: Crear la variable

$$\mathbf{y} = [-15 - 12] + \mathbf{x}$$

donde $\mathbf{x}$ es el vector del ejemplo anterior



2.5.1. Comandos básicos de gestión del workspace

- `whos`

  Lista las variables actualmente vigentes del workspace

```
>> whos
  Name Size Bytes  Class
  B     2x2    64  double array (complex)
  M     4x3    96  double array
  V     4x1    32  double array
  x     1x2    16  double array
  y     1x2    16  double array
Grand total is 24 elements using 224 bytes
```

- `whos var1 var2 ...`

  Solo lista las variables especificadas

```
>> whos B M
  Name Size Bytes  Class
  B     2x2    64  double array (complex)
  M     4x3    96  double array
Grand total is 16 elements using 160 bytes
```

- `clear var1 var2 ...`

  Borra solo las variables especificadas (var1, var2, ...)

```
>> clear M V
>> whos
  Name Size Bytes  Class
  B     2x2    64  double array (complex)
```

```
  x     1x2    16  double array
  y     1x2    16  double array
Grand total is 8 elements using 96 bytes
>>
```

- `clear`

  Borra todas las variables del workspace

```
>> clear
>> whos
>>
```

Observación:

- Una vez eliminado el contenido entero del workspace, una llamada a whos, nos retorna de inmediato el prompt, pues no hay nada que listar.

- Una vez que una variable ha sido borrada del workspace, ésta no será posible de volver a recuperar.

2.6. Palabras reservadas

Las palabras reservadas (keywords) son aquellas que son de uso exclusivo del interprete de MATLAB. El listado de las palabras reservadas (20) lo obtenemos mediante el comando **iskeyword**.

```
>> iskeyword
ans =
    'break'
    'case'
    'catch'
    'classdef'
    'continue'
    'else'
    'elseif'
    'end'
    'for'
    'function'
    'global'
    'if'
    'otherwise'
    'parfor'
    'persistent'
    'return'
    'spmd'
    'switch'
    'try'
    'while'
```

2.7. Comandos especiales

MATLAB posee funciones (comandos) que retornan valores de gran importancia, los cuales son utilizados a menudo en la mayoría de programas MATLAB.

Función	Valor retornado
ans	Retorna el valor de salida de alguna expresión que no ha sido asignada a alguna variable.
eps	Precisión relativa de punto flotante
intmax	Entero mas grande que la computadora puede representar.
intmin	Entero mas pequeño que la computadora puede representar.
realmax	Numero de Punto Flotante mas grande que la computadora puede representar.
realmin	Numero de Punto Flotante mas pequeño que la computadora puede representar.
pi	3.1415926535897...
i, j	Unidad imaginaria.
Inf	Infinito (n/0).
NaN	Not a Number. (representa una indeterminación: 0/0 , inf/inf, inf-inf, ...).
computer	Tipo de computadora.
version	Cadena con la versión de MATLAB.

2.8. Las Funciones Internas MATLAB

La funciones provistas por MATLAB (funciones internas) pueden ser:

- Funciones de archivo M, aquellas que son implementadas como archivos M.
- Funciones built-ins, aquellas que son programas ejecutables precompilados.

Muchas de las funciones MATLAB internas están sobrecargadas, de manera que puedan manipular diferentes tipos de dato eficientemente. Las funciones internas de MATLAB se encuentran en los subdirectorios del directorio toolbox\matlab

```
>> dir([matlabroot '\toolbox\matlab'])

.               demos           helptools       plottools       system
..              elfun           icons           plugins         testframework
apps            elmat           imagesci        polyfun         timefun
audiovideo      funfun          iofun           randfun         timeseries
codetools       general         lang            scribe          uitools
configtools     graph2d         matfun          settings        verctrl
connector       graph3d         matlab.settings sparfun         winfun
datafun         graphics        mcc.enc         specfun         datamanager
guide           ops             specgraph       datatypes       hds
optimfun        strfun
```

Para listar las funciones de cada subdirectorio (categoría) y poder acceder a la documentación de cada una de las funciones que éstas contienen, digitamos doc o help seguido del nombre del subdirectorio desde el prompt de la ventana comandos.

Observación: A diferencia de las funciones de archivo M, las funciones built-ins no permiten ver su código fuente; sin embargo, la mayoría de éstas funciones tienen un archivo M asociado a ellas, el cual solo contiene documentación de ayuda para la función.

2.9. Las Expresiones y los Operadores

2.9.1. Las Expresiones

Las expresiones están constituidas por la combinación de operadores aritméticos, relacionales y lógicos aplicados sobre operandos.

1. **Expresión Unaria**

```
OPERANDO Operador
```

2. **Expresión Binaria**

```
Operador1 OPERANDO Operador2
```

En MATLAB, las expresiones son evaluadas de izquierda a derecha. Cuando las expresiones son evaluadas se sigue la regla de precedencia de operadores MATLAB:

- Paréntesis ()
- Transpuesta (.'), potenciación (.^), transpuesta conjugada ('), potenciación matricial (^)
- Mas unario (+), menos unario (-), negación lógica (~).
- Multiplicación (.*), división derecha (./), división izquierda (.\), mutiplicación matricial (*), división derecha matricial (/), división izquierda matricial (\)
- Adición (+), sustracción (-)
- Operador dos puntos (:)
- Menor que (<), menor o igual (<=), mayor que (>), mayor o igual que (>=), idéntico a (==), diferente de (~=)
- AND elemento a elemento (&)
- OR elemento a elemento (|)
- AND en corto circuito (&&)
- OR en corto circuito (||)

2.9.2. Los Operadores Aritméticos

Las operaciones aritméticas se determinan dependiendo de la concepción que se tenga sobre el arreglo. MATLAB permite concebir una disposición de valores de una misma clase en filas y columnas como arreglo o matriz.

- Como arreglo, las operaciones aritméticas serán elemento a elemento (elementwise);
- Como matriz, las operaciones aritméticas son las basadas en reglas del álgebra lineal.

OPERACIÓN	TIPO	
	MATRIZ	ARREGLO
Adición	+	+
Sustracción	-	-
Multiplicación	*	.*
División Izquierda	\	.\
División Derecha	/	./
Exponenciación	^	.^

Los Operadores Aritméticos del Tipo Arreglo (elemento a elemento)

OPERACIÓN TIPO ARREGLO (elemento a elemento)		REGLA DE CORRESPONDENCIA
+/-	$C_{m,n} = A_{m,n} \pm B_{m,n}$	$c_{ij} = a_{ij} \pm b_{ij}$
.*	$C_{m,n} = A_{m,n} .* B_{m,n}$	$c_{ij} = a_{ij} \cdot b_{ij}$
./	$C_{m,n} = A_{m,n} ./ B_{m,n}$	$c_{ij} = a_{ij} \cdot b_{ij}^{-1} = \frac{a_{ij}}{b_{ij}}$
.\	$C_{m,n} = A_{m,n} .\backslash B_{m,n}$	$c_{ij} = a_{ij}^{-1} \cdot b_{ij} = \frac{b_{ij}}{a_{ij}}$
.^	$C_{m,n} = A_{m,n} .\hat{} B_{m,n}$	$c_{ij} = a_{ij}^{b_{ij}}$

Los Operadores Aritméticos del Tipo Matriz (álgebra lineal)

OPERACIÓN TIPO MATRIZ (reglas del álgebra lineal)		REGLA DE CORRESPONDENCIA
+/-	$C_{m,n} = A_{m,n} \pm B_{m,n}$	$c_{ij} = a_{ij} \pm b_{ij}$
*	$C_{m,p} = A_{m,n} * B_{n,p}$	$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$
/	$C_{m,n} = A_{m,n} / B_{n,n}$	$C_{m,n} = A_{m,n} B_{n,n}^{-1}$
\	$C_{m,n} = A_{m,m} \backslash B_{m,n}$	$C_{m,n} = A_{m,m}^{-1} B_{m,n}$
^	$C_{m,m} = A_{m,m} \hat{} p$ (p: escalar)	$C_{m,m} = \underbrace{A_{m,m} A_{m,m} \cdots A_{m,m}}_{p \text{ veces}}$

Funciones Equivalentes

Todas estas operaciones tienen sus equivalentes en forma de funciones MATLAB Internas.

OPERACIÓN ARITMÉTICA	EXPRESIÓN	FUNCIÓN EQUIVALENTE
Adición binaria	`A+B`	`plus(A,B)`
Mas unario	`+A`	`uplus(A)`
Sustracción binaria	`A-B`	`minus(A,B)`
Menos unario	`-A`	`unminus(A)`
Multiplicación matricial	`A*B`	`mtimes(A,B)`
Multiplicación de arreglos	`A.*B`	`times(A,B)`
División derecha matricial	`A/B`	`mrdivide(A,B)`
División derecha de arreglos	`A./B`	`rdivide(A,B)`
División izquierda matricial	`A\B`	`mldivide(A,B)`
División izquierda de arreglos	`A.\B`	`ldivide(A,B)`
Potenciación matricial	`A^B`	`mpower(A,B)`
Potenciación de arreglos	`A.^B`	`power(A,B)`
Transpuesta compleja	`A'`	`ctranspose(A)`
Transpuesta matricial	`A.'`	`transpose(A)`

2.9.3. Los Operadores Relacionales

Los operadores relacionales comparan los operandos cuantitativamente, usando los siguientes operadores. Realizan las comparaciones elemento a elemento entre los operandos. Retornan un arreglo de la clase logical de la dimensión de los operandos:

- 1 lógico (true) : si la relación es verdadera
- 0 lógico (false) : si la relación es falsa.

Operador	Descripción
`<`	Menor que
`>`	Mayor que
`<=`	Menor o igual que
`>=`	Mayor o igual que
`==`	Igual a
`~=`	Diferente de

2.9.4. Los Operadores Lógicos

Los hay de tres tipos:

- **Elemento a Elemento**

Operador	Descripción
`&`	Retorna 1 lógico (true) en caso sean verdaderos ambos elementos de las mismas posiciones en los arreglos; en caso contrario retorna 0 lógico (false).
`|`	Retorna 0 lógico (false) en caso sean falsos ambos elementos de las mismas posiciones en los arreglos; en caso contrario retorna 1 lógico (true).
`~`	Complementa cada elemento del arreglo
`xor`	Retorna 1 (lógico) en caso sean verdaderos un elemento y falso el otro elemento cuyas posiciones en los arreglos sea la misma; en caso contrario retorna 0 lógico (false).

- **Bit a Bit (bitwise)**

  Compara cantidades binarias, bit a bit: bitand, bitor, bitcmp y bitxor.

- **Corto Circuito**

  Evaluan el segundo operando solo cuando el resultado no quede completamente determinado por la evaluación del primer operando.

Operador	Descripción
`&&`	Retorna 1 lógico (true) si ambas entradas son verdaderas; y el 0 lógico si alguna de ellas no lo es.
`||`	Retorna 1 lógico (true) si una o ambas entradas son verdaderas; y el 0 lógico si ambas no lo son.

2.10. La Indexación de Matrices

2.10.1. Los Vectores Rango

Permiten generar vectores fila a través de una progresión aritmética. Se pueden crear de dos formas:

- `vi:vf`

  Genera una secuencia numérica iniciando en vi e incrementándose en +1 unidades hasta llegar a vf.

```
>> t = 2008:2011
t =
    2008    2009    2010    2011
```

- `vi:step:vf`

  Genera una secuencia numérica iniciando en vi e incrementándose en step unidades hasta vf.

```
>> t = 2008:3:2014
t =
    2008    2011    2014

>> t = 2008:3:2018
t =
    2008    2011    2014    2017
```

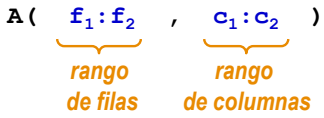
Observación: En caso algún rango sea inconsistente, MATLAB generará como resultado una matriz vacía (1x0).

2.10.2. La Indexación Bidimensional

Dada la matriz A de $m \times n$

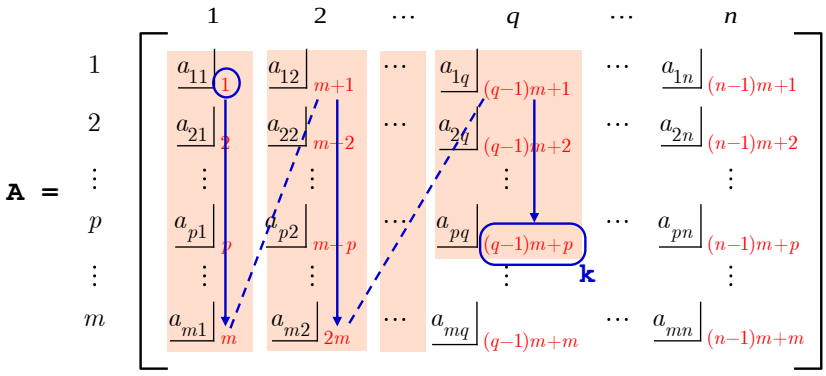
$$\mathbf{A} = \begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ f_1 \\ \vdots \\ f_2 \\ \vdots \\ m \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & \cdots & c_1 & \cdots & c_2 & \cdots & n \end{array} \\ \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{1,c_1} & \cdots & a_{1,c_2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,c_1} & \cdots & a_{2,c_2} & \cdots & a_{2,n} \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ a_{f_1,1} & a_{f_1,2} & \cdots & a_{f_1,c_1} & \cdots & a_{f_1,c_2} & \cdots & a_{f_1,n} \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ a_{f_2,1} & a_{f_2,2} & \cdots & a_{f_2,c_1} & \cdots & a_{f_2,c_2} & \cdots & a_{f_2,n} \\ \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,c_1} & \cdots & a_{m,c_2} & \cdots & a_{m,n} \end{bmatrix} \end{array}$$

Para hacer referencia (indexar) a los elementos comprendidos entre las filas f_1 y f_2 y columnas c_1 y c_2 mediante el uso de dos índices rango lo hacemos así



2.10.3. La Indexación Lineal

La indexación lineal, se lleva acabo cuando se desea hacer referencia a un elemento de una matriz mediante un solo índice.



Para llevarlo a cabo utilizamos

```
A( 1:k )
```

donde `k` hace referencia al elemento de índice (`p`,`q`) y se relaciona mediante `k = (q-1)m + p`

2.10.4. La Indexación Lógica

Se utiliza como índice una matriz con elementos de la clase logical (B), de la misma dimensión que la de la matriz por indexar (A).

A(B)

El resultado de la indexación, será el listado (en forma de vector columna) de los elementos de la matriz A, cuyos valores lógicos respectivos (de la misma posición en fila y columna) en la matriz B sean 1 lógico.

EJEMPLO: Sean las matrices A y B que se indican (B matriz de valores lógicos):

$$A = \begin{bmatrix} 12 & 32 & 11 & 4 \\ 11 & 3 & 3 & 34 \\ 23 & 23 & 45 & 2 \\ 45 & 17 & 23 & 1 \end{bmatrix} \qquad B = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}}_{\text{valores lógicos}}$$

Entonces:

```
>> E = A(B)
E =
   45
    3
   45
   23
   34
```

Observación: La búsqueda se hace por columnas y el resultado es devuelto en forma de vector columna

2.10.5. El operador :

Permite hacer referencia a todo un rango de fila o de columna, según donde sea especificado.

EJEMPLO: Sean la matriz R :

$$R = \begin{pmatrix} 47 & 58 & 69 & 80 & 1 \\ 57 & 68 & 79 & 9 & 11 \\ 67 & 78 & 8 & 10 & 21 \\ 77 & 7 & 18 & 20 & 31 \\ 6 & 17 & 19 & 30 & 41 \\ 16 & 27 & 29 & 40 & 51 \\ 26 & 28 & 39 & 50 & 61 \\ 36 & 38 & 49 & 60 & 71 \\ 37 & 48 & 59 & 70 & 81 \end{pmatrix}$$

Entonces:

```
>> A = R( 2:5 , : )
A =
   57   68   79    9   11
   67   78    8   10   21
   77    7   18   20   31
    6   17   19   30   41

>> B = R( : , 3:5 )
B =
   69   80    1
   79    9   11
    8   10   21
   18   20   31
   19   30   41
   29   40   51
   39   50   61
   49   60   71
   59   70   81

>> C = R( 7 , : )
C =
   26   28   39   50   61
```

2.10.6. La palabra reservada end

La palabra reservada end, al utilizarse en indexación, indica el último índice del rango posible de la dimensión (fila ó columna) en la que aparezca.

EJEMPLO: Sean la matriz R :

$$R = \begin{pmatrix} 47 & 58 & 69 & 80 & 1 \\ 57 & 68 & 79 & 9 & 11 \\ 67 & 78 & 8 & 10 & 21 \\ 77 & 7 & 18 & 20 & 31 \\ 6 & 17 & 19 & 30 & 41 \\ 16 & 27 & 29 & 40 & 51 \\ 26 & 28 & 39 & 50 & 61 \\ 36 & 38 & 49 & 60 & 71 \\ 37 & 48 & 59 & 70 & 81 \end{pmatrix}$$

Entonces:

```
>> D = R(6:9,end)
D =
  51   61   71   81

>> D = R(end,:)
D =
  37   48   59   70   81

>> D = R(end,end-1)
D =
  70

>> D = R(end,end)
D =
  81

>> D = R(end-1)
D =
  71

>> D = R(end)
D =
  81
```

2.11. Gestión de Archivos en MATLAB

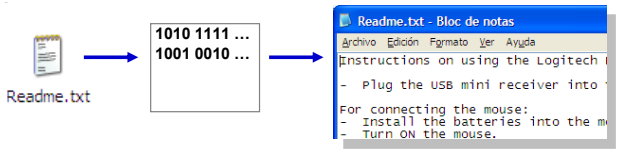
2.11.1. Los Tipos de Archivo soportados por MATLAB

Todos los datos almacenados en el sistema MATLAB (workspace) son binarios, es decir descritos con ceros y unos; comúnmente suelen almacenarse en archivos, los cuales se clasifican en dos grandes grupos:

- **Archivos de Texto**

  Son aquellos cuyos códigos binarios son interpretados directamente como caracteres (letras, dígitos y/o símbolos) especificados en código UNICODE (extensión del ASCII) por cualquier editor universal del texto.

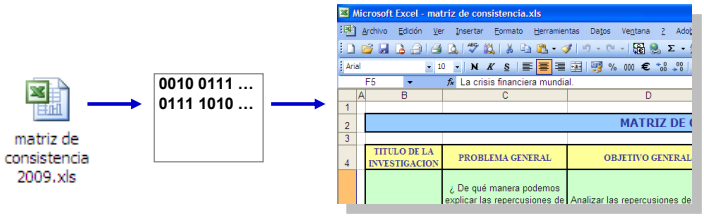
  Por ejemplo: .txt, .m, .dyn, .tex, etc.



- **Archivos Binarios**

  Son aquellos cuyos códigos binarios son interpretados según el programa en el que fueron generados; por lo general, no tienen interpretación en forma de texto.

  Por ejemplo: .mat, .jpg, .xls, etc.

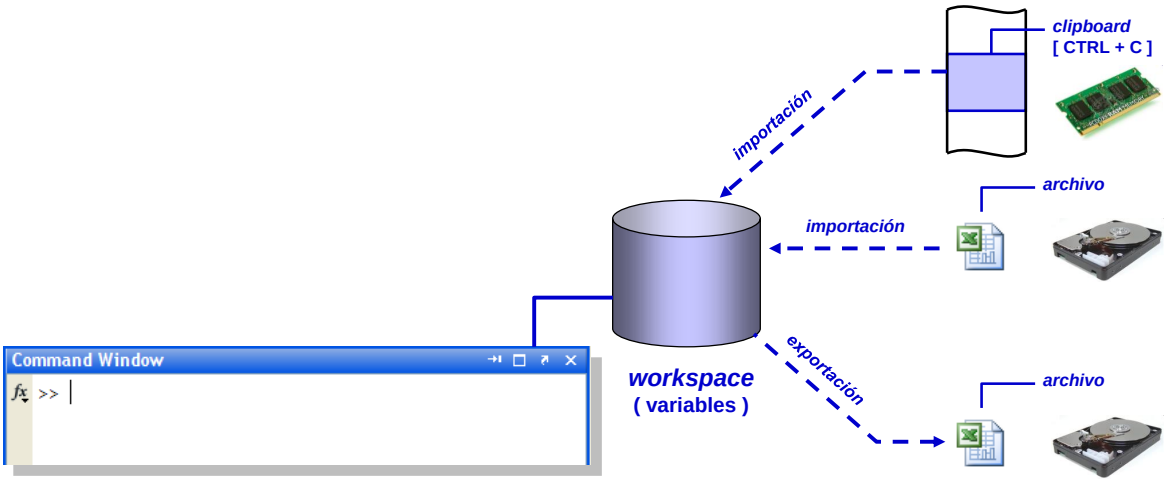


Los principales tipos de datos soportados por MATLAB se presentan en la siguiente tabla.

Tipo de Archivo	Extensión	Descripción	Función de Importación	Función de Exportación
Texto	cualquiera	Numeros delimitados con espacio en blanco	`load`	`save -ascii`
		Numeros delimitados	`dlmread`	`dlmwrite`
		Numeros delimitados con comas	`csvread`	`csvwrite`
		Cualquier de los formatos previos, o una mixtura de cadenas y numeros	`textscan`	
Dato con formato MATLAB (binario MATLAB)	`MAT`	Variable(s) almacenada(s) en el Workspace de MATLAB	`load`	`save`
Hoja de cálculo	`XLS`	Hoja de Cálculo Microsoft Excel	`xlsread`	`xlswrite`
	`XLSX` `XLSB` `XLSM`	Formatos soportados con Excel® 2007		
	`WK1`	Formato soportado por Lotus 1-2-3	`wk1read`	`wk1write`
Extended Markup Language	`XML`	Texto con formato XML	`xmlread`	`xmlwrite`
Dato Científico	`CDF`	Formato de Dato Común	`cdfread`	`cdfwrite`
	`FITS`	Flexible Image Transport System	`fitsread`	`none`
	`HDF`	Hierarchical Data Format, version 4, o HDF-EOS v. 2	`hdfread`	
	`H5`	HDF o HDF-EOS, version 5	`hdf5read`	`hdf5write`
	`NC`	Network Common Data Form (netCDF)	`netcdf`	`netcdf`

2.11.2. Importación y Exportación de Datos en MATLAB

La importación de datos es el proceso que permite cargar datos desde archivos de disco o del clipboard a variables del workspace, mientras que la exportación de datos es el proceso que permite almacenar variables del workspace a archivos de disco.



Observaciones:

- Los mecanismos de importación o exportación dependen de los datos a transferirse.
- El conjunto de funciones MATLAB que permiten realizar la Exportación e Importación de datos-frecuentemente se les denominan funciones I/O de alto nivel.
- Para el caso de datos que no sean soportados por las funciones I/O de alto nivel se cuenta con lasfunciones I/O de bajo nivel las cuales están basadas en la Biblioteca ANSI del C Estándar.

Importación y Exportación de Datos en Formato Texto

La importación o exportación en formato texto se efectúa considerando por cada archivo una sola variable.

- **Importación**
  - `load nombrearchivo`
  - `load -ascii nombrearchivo`
    Carga el archivo en una variable del workspace con nombre nombrearchivo. El archivo debe contener los números separados por un caracter espacio en blanco y distribuidos en forma matricial, separando las filas con un cambio de línea.
  - `mivariable = load('nombrearchivo')`
    Carga el archivo en una variable con el nombre especificado en mivariable
  - `mivariable = dlmread('nombrearchivo' , strDelimitador)`
    Carga el archivo en una variable con el nombre especificado en mivariable especificando en la cadena strDelimitador, el caracter de separación utilizado entre los números.
- **Exportación**
  - `save nombrearchivo variable -ascii`
    Guarda el contenido de la variable en el archivo nombrearchivo en formato numérico separando los elementos en las filas por un caracter espacio en blanco.
  - `dlmwrite('nombrearchivo', variable, strDelimitador)`
    Guarda el contenido de la variable en el archivo nombrearchivo en formato numérico delimitando los elementos en las filas con el carácter especificado en strDelimitador.

Importación y Exportación de Datos en Formato MATLAB

La importación o exportación en formato binario MATLAB (doble precisión) se efectúa considerando por cada archivo una o mas variables.

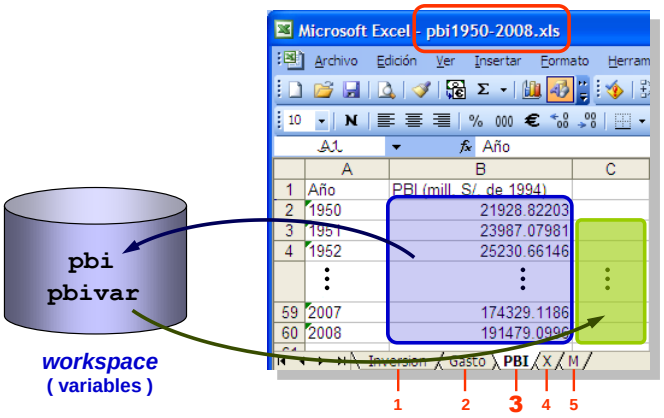
- **Importación**
  - `load nombrearchivo`
    Carga todas las variables contenidas en nombrearchivo.mat al workspace. Si el archivo no tiene formato binario MATLAB, lo tratará como texto.
  - `load nombrearchivo var1 var2 ...`
    Carga las variables var1 var2 ... contenidas en nombrearchivo
- **Exportación**
  - `save nombrearchivo`
    Guarda todas las variables contenidas en el workspace en el archivo nombrearchivo.mat
  - `save nombrearchivo var1 var2 ...`
    Carga las variables var1 var2 ... en el archivo nombrearchivo.mat

Importación y Exportación de Datos en Formato Excel

- **Importación**
  - `variable = xlsread('nombrearchivo', numHoja, strRango)`
    Carga en variable el contenido en el rango strRango, ubicado en la hoja numHoja, del archivo nombrearchivo.xls

- **Exportación**
  - `xlswrite('nombrearchivo', variable, numHoja, strCeldaInicial)`
    Guarda el contenido de variable a partir de la celda strCelda, ubicada en la hoja numHoja, del archivo nombrearchivo.xlswrite

EJEMPLO: Analizar las siguientes comandos




```
>> pbi = xlsread('pbi1950-2008',3,'B2:B60');
>> pbivar = diff(pbi)./pbi(1:end-1);
>> xlswrite('pbi1950-2008',pbivar,3,'C3');
```

2.11.3. Generación de Sentencias LaTeX a partir de variables MATLAB

Para representar el contenido de una variable MATLAB en formato LaTeX:

1. Se convierte a formato simbólico el contenido de alguna variable numérica MATLAB aplicando la función sym.

```
>> M = magic(5)
M =
  17   24    1    8   15
  23    5    7   14   16
   4    6   13   20   22
  10   12   19   21    3
  11   18   25    2    9

>> x = sym(M);
```

2. Se aplica el comando latex sobre el objeto simbólico que representa a la variable numérica obteniéndose la sentencia LaTeX como respuesta.

```
>> eq1 = latex(x)
eq1 =
   \left(\begin{array}{ccccc} 17 & 24 & 1 & 8 & 15\\ 23 & 5 & 7 & 14 & 16\\
    4 & 6 & 13 & 20 & 22\\ 10 & 12 & 19 & 21 & 3\\ 11 & 18 & 25 & 2 & 9
   \end{array}\right)
```

EJERCICIOS DEL CAPITULO 2

1. Elementos del Lenguaje

1. Sean los arreglos

$$A = \begin{bmatrix} -3 & 4 & -2 & 0 \\ 1 & -2 & 3 & 7 \\ 5 & 0 & 0 & -2 \end{bmatrix} \quad B = \begin{bmatrix} 4 \\ -9 \\ 2 \\ 3 \end{bmatrix} \quad C = \begin{bmatrix} 2 & 1 & 4 & 5 \end{bmatrix}$$

a) Ingréselos al workspace base

```
>> A = [-3 4 -2 0; 1 -2 3 7; 5 0 0 -2]
A =
    -3     4    -2     0
     1    -2     3     7
     5     0     0    -2

>> B = [4; -9; 2; 3]
B =
     4
    -9
     2
     3
>> C = [2 1 4 5]
C =
     2     1     4     5
```

b) Construya el arreglo

$$D = \left[\begin{array}{c|c} A' & B \\ \hline C & \end{array}\right]$$

```
>> D = [ A.' B; C ]
D =
    -3     1     5     4
     4    -2     0    -9
    -2     3     0     2
     0     7    -2     3
     2     1     4     5
```

c) Construya el arreglo

$$E = \left[\begin{array}{c|c} A & \\ \hline C & B \end{array}\right]$$

```
>> E = [ [ A; C] B ]
E =
    -3     4    -2     0     4
     1    -2     3     7    -9
     5     0     0    -2     2
     2     1     4     5     3
```

2. Liste las variables contenidas en el workspace base.

```
>> whos
Name Size Bytes Class   Attributes
A     3x4    96 double
B     4x1    32 double
C     1x4    32 double
D     5x4   160 double
E     4x5   160 double
```

3. Dadas las cadenas (string) de texto
str1: *"En momentos de crisis"*
str2: *"usa tu imaginación"*
construya la cadena
str3: *"En momentos de crisis, usa tu imaginacion"*

```
>> str1 = 'En momentos de crisis';
>> str2 = 'usa tu imaginacion';
>> str3 = [ str1 ', ' str2 ]
str3 =
En momentos de crisis, usa tu imaginacion
```

4. En una variable V se tiene almacenado el valor numérico 221.4127 el cual representa un VAN en unidades Nuevos Soles.
Construir una cadena de texto que visualice el mensaje
"VAN: 221.4127 Nuevos Soles"

```
>> V = 221.4127;
>> str = ['VAN: ' num2str(V) ' Nuevos Soles']
str =
VAN: 221.4127 Nuevos Soles
```

2. Funciones MATLAB Internas

5. Imprima el directorio raíz de MATLAB

```
>> matlabroot
ans =
C:\Archivos de programa\MATLAB\R2013a
```

6. Imprima el directorio actual de trabajo de MATLAB

```
>> pwd
ans =
C:\Documents and Settings\Admin\Mis documentos\MATLAB
```

7. Imprima las rutas de búsqueda de funciones de MATLAB

```
>> matlabpath
MATLABPATH
C:\Documents and Settings\Miguel\Mis documentos\MATLAB
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\general
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\ops
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\lang
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\elmat
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\elfun
C:\Archivos de programa\MATLAB\R2007b\toolbox\matlab\specfun
:
```

8. Abra el Explorador de Windows visualizando el contenido del directorio actual de trabajo de MATLAB.

```
>> !explorer .
```

9. Liste los directorios en los que se hallan las funciones provistas por MATLAB (funciones MATLAB internas)

```
>> strFuncInter = [matlabroot '\toolbox\matlab'];
>> dir(strFuncInter)

.                graph3d          polyfun
..               graphics         randfun
apps             guide            scribe
audiovideo       hds              settings
codetools        helptools        sparfun
configtools      icons            specfun
connector        imagesci         specgraph
datafun          iofun            strfun
datamanager      lang             system
datatypes        matfun           timefun
demos            matlab.settings  timeseries
elfun            mcc.enc          uitools
elmat            ops              verctrl
funfun           optimfun         winfun
general          plottools
graph2d          plugins
```

10. Liste las funciones provistas por MATLAB para el análisis de datos

```
>> help datafun
  Data analysis and Fourier transforms.

  Basic operations.
    max         - Largest component.
    min         - Smallest component.
    mean        - Average or mean value.
    median      - Median value.
    mode        - Mode, or most frequent value in a sample.
    std         - Standard deviation.
    var         - Variance.
    sort        - Sort in ascending order.
    sortrows    - Sort rows in ascending order.
    issorted    - TRUE for sorted vector and matrices.
    sum         - Sum of elements.
    prod        - Product of elements.
    hist        - Histogram.
    histc       - Histogram count.
    trapz       - Trapezoidal numerical integration.
    cumsum      - Cumulative sum of elements.
    cumprod     - Cumulative product of elements.
    cumtrapz    - Cumulative trapezoidal numerical
                  integration.

  Finite differences.
    diff        - Difference and approximate derivative.
    gradient    - Approximate gradient.
    del2        - Discrete Laplacian.
```

```
  Correlation.
    corrcoef    - Correlation coefficients.
    cov         - Covariance matrix.
    subspace    - Angle between subspaces.

  Filtering and convolution.
    filter      - One-dimensional digital filter.
    filter2     - Two-dimensional digital filter.
    conv        - Convolution and polynomial multiplication.
    conv2       - Two-dimensional convolution.
    convn       - N-dimensional convolution.
    deconv      - Deconvolution and polynomial division.
    detrend     - Linear trend removal.

  Fourier transforms.
    fft         - Discrete Fourier transform.
    fft2        - Two-dimensional discrete Fourier transform.
    fftn        - N-dimensional discrete Fourier Transform.
    ifft        - Inverse discrete Fourier transform.
    ifft2       - Two-dimensional inverse discrete Fourier
                  transform.
    ifftn       - N-dimensional inverse discrete Fourier
                  Transform.
    fftshift    - Shift zero-frequency component to center
                  of spectrum.
    ifftshift   - Inverse FFTSHIFT.
    fftw        - Interface to FFTW library run-time
                  algorithm tuning control.
```

11. Visualice la ayuda en formato texto y HTML de la función **filter**

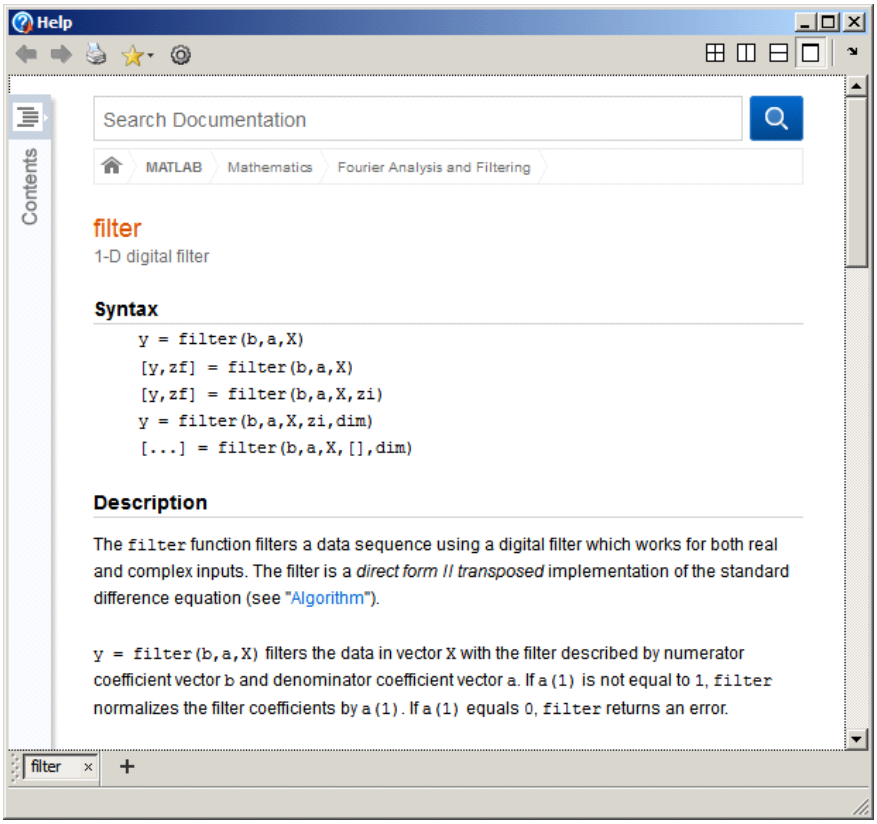
a) Formato Texto (Ventana de Comando)

```
>> help filter
 filter One-dimensional digital filter.
    Y = filter(B,A,X) filters the data in vector X with the
    filter described by vectors A and B to create the filtered
    data Y.  The filter is a "Direct Form II Transposed"
    implementation of the standard difference equation:

    a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)
                          - a(2)*y(n-1) - ... - a(na+1)*y(n-na)

    If a(1) is not equal to 1, filter normalizes the filter
    coefficients by a(1).

    filter always operates along the first non-singleton dimension,
    namely dimension 1 for column vectors and non-trivial matrices,
    and dimension 2 for row vectors.

    [Y,Zf] = filter(B,A,X,Zi) gives access to initial and final
    conditions, Zi and Zf, of the delays.  Zi is a vector of length
    MAX(LENGTH(A),LENGTH(B))-1, or an array with the leading dimension
    of size MAX(LENGTH(A),LENGTH(B))-1 and with remaining dimensions
    matching those of X.

    filter(B,A,X,[],DIM) or filter(B,A,X,Zi,DIM) operates along the
```

```
    dimension DIM.

    See also filter2, filtfilt, filtic.

    Note: FILTFILT and FILTIC are in the Signal Processing Toolbox.

    Overloaded methods:
       gjr/filter
       garch/filter
       egarch/filter
       arima/filter
       LagOp/filter
       fints/filter
       timeseries/filter
       SigLogSelector.filter
       gpuArray/filter
       fxptui.filter
       dfilt.filter
       gf/filter
       channel.filter
       mfilt.filter
       adaptfilt.filter
       sweepsetfilter/filter
       sweepset/filter

    Reference page in Help browser
       doc filter
```

b) Formato HTML (Help Browser)

```
>> doc filter
```



12. Liste las funciones provistas por MATLAB para la manipulación de matrices y construcción de matrices elementales (elmat); luego, apoyándose en la información obtenida, obtenga en las variables m y n el número de filas y columnas de A; en p el número de filas de P y en q el número de columnas de Q

$$P = [D \;\mid\; E'] \quad Q = \left[P' \left| \frac{C'}{B} \right. \right]$$

```
>> [m,n] = size(A)
m =
     3
n =
     4

>> P = [D E.']
P =
    -3     1     5     4    -3     1     5     2
     4    -2     0    -9     4    -2     0     1
    -2     3     0     2    -2     3     0     4
     0     7    -2     3     0     7    -2     5
     2     1     4     5     4    -9     2     3

>> p = size(P,1)
p =
     5

>> Q = [ P.' [ C.'; B] ]
Q =
    -3     4    -2     0     2     2
     1    -2     3     7     1     1
     5     0     0    -2     4     4
     4    -9     2     3     5     5
    -3     4    -2     0     4     4
     1    -2     3     7    -9    -9
     5     0     0    -2     2     2
     2     1     4     5     3     3

>> q = size(Q,2)
q =
     6
```

13. Obtenga el valor del máximo y mínimo elemento, por columna, del arreglo Q así como la posición (fila) en que tales elementos se encuentran.

```
>> [maxQ,maxInd] = max(Q)
maxQ =
     5     4     4     7     5     5
maxInd =
     3     1     8     2     4     4

>> [minQ,minInd] = min(Q)
minQ =
    -3    -9    -2    -2    -9    -9
minInd =
     1     4     1     3     6     6
```

14. Obtenga el valor de la suma, producto, promedio, varianza y desviación estándar muestral, por columna, del arreglo Q.

```
>> SQ = sum(Q)
SQ =
    12    -4     8    18    12    12
>> PQ = prod(Q)
PQ =
        1800     0      0      0   -8640       -8640
>> mQ = mean(Q)
mQ =
    1.5000   -0.5000    1.0000    2.2500    1.5000    1.5000
>> vQ = var(Q)
vQ =
   10.2857   17.1429    5.4286   14.2143   19.7143   19.7143
>> sQ = std(Q)
sQ =
    3.2071    4.1404    2.3299    3.7702    4.4401    4.4401
```

15. Construya los arreglos S y T de tal manera que cada una de sus columnas estén constituidas por los elementos de las columnas de Q ordenados ascendente y descendentemente respectivamente.

```
>> S = sort(Q,'ascend')
S =
    -3    -9    -2    -2    -9    -9
    -3    -2    -2    -2     1     1
     1    -2     0     0     2     2
     1     0     0     0     2     2
     2     0     2     3     3     3
     4     1     3     5     4     4
     5     4     3     7     4     4
     5     4     4     7     5     5
>> T = sort(Q,'descend')
T =
     5     4     4     7     5     5
     5     4     3     7     4     4
     4     1     3     5     4     4
     2     0     2     3     3     3
     1     0     0     0     2     2
     1    -2     0     0     2     2
    -3    -2    -2    -2     1     1
    -3    -9    -2    -2    -9    -9
```

16. Obtenga los índices en los que inicialmente se ubicaban los elementos

```
>> [S,posS] = sort(Q,'ascend')
S =
    -3    -9    -2    -2    -9    -9
    -3    -2    -2    -2     1     1
     1    -2     0     0     2     2
     1     0     0     0     2     2
     2     0     2     3     3     3
     4     1     3     5     4     4
     5     4     3     7     4     4
     5     4     4     7     5     5
posS =
     1     4     1     3     6     6
     5     2     5     7     2     2
     2     6     3     1     1     1
```

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

>> [T,posT] = sort(Q,'descend')
T =
     5     4     4     7     5     5
     5     4     3     7     4     4
     4     1     3     5     4     4
     2     0     2     3     3     3
     1     0     0     0     2     2
     1    -2     0     0     2     2
    -3    -2    -2    -2     1     1
    -3    -9    -2    -2    -9    -9
posT =
     3     1     8     2     4     4
     7     5     2     6     3     3
     4     8     6     8     5     5
     8     3     4     4     8     8
     2     7     3     1     1     1
     6     2     7     5     7     7
     1     6     1     3     2     2
     5     4     5     7     6     6
```

17. Construya una vector **x** de 100000 muestras con media 0,6 y desviación estándar 0,2. Visualice el resultado en formato bancario (2 dígitos decimales).

```
>> format bank

>> E = randn(100000,1);

>> mE = mean(E), dE = std(E)
mE =
         -0.00
dE =
          1.00

>> x = 0.6 + 0.2*E;

>> mx = mean(x), dx = std(x)
mx =
          0.60
dx =
          0.20
```

18. Liste las funciones matemáticas elementales provistas por MATLAB

```
>> help elfun
  Elementary math functions.

  Trigonometric.
    sin         - Sine.
    sind        - Sine of argument in degrees.
    sinh        - Hyperbolic sine.
    asin        - Inverse sine.
    asind       - Inverse sine, result in degrees.
```

```
  asinh       - Inverse hyperbolic sine.
  cos         - Cosine.
  cosd        - Cosine of argument in degrees.
  cosh        - Hyperbolic cosine.
  acos        - Inverse cosine.
  acosd       - Inverse cosine, result in degrees.
  acosh       - Inverse hyperbolic cosine.
  tan         - Tangent.
  tand        - Tangent of argument in degrees.
  tanh        - Hyperbolic tangent.
  atan        - Inverse tangent.
  atand       - Inverse tangent, result in degrees.
  atan2       - Four quadrant inverse tangent.
  atan2d      - Four quadrant inverse tangent, result
                in degrees.
  atanh       - Inverse hyperbolic tangent.
  sec         - Secant.
  secd        - Secant of argument in degrees.
  sech        - Hyperbolic secant.
  asec        - Inverse secant.
  asecd       - Inverse secant, result in degrees.
  asech       - Inverse hyperbolic secant.
  csc         - Cosecant.
  cscd        - Cosecant of argument in degrees.
  csch        - Hyperbolic cosecant.
  acsc        - Inverse cosecant.
  acscd       - Inverse cosecant, result in degrees.
  acsch       - Inverse hyperbolic cosecant.
  cot         - Cotangent.
  cotd        - Cotangent of argument in degrees.
  coth        - Hyperbolic cotangent.
  acot        - Inverse cotangent.
  acotd       - Inverse cotangent, result in degrees.
  acoth       - Inverse hyperbolic cotangent.
  hypot       - Square root of sum of squares.

Exponential.
  exp         - Exponential.
  expm1       - Compute exp(x)-1 accurately.
  log         - Natural logarithm.
  log1p       - Compute log(1+x) accurately.
  log10       - Common (base 10) logarithm.
  log2        - Base 2 logarithm and dissect floating
                point number.
  pow2        - Base 2 power and scale floating point
                number.
  realpow     - Power that will error out on complex result.
  reallog     - Natural logarithm of real number.
  realsqrt    - Square root of number greater than or equal
                to zero.
  sqrt        - Square root.
  nthroot     - Real n-th root of real numbers.
  nextpow2    - Next higher power of 2.

Complex.
  abs         - Absolute value.
  angle       - Phase angle.
  complex     - Construct complex data from real and
```

```
                imaginary parts.
    conj        - Complex conjugate.
    imag        - Complex imaginary part.
    real        - Complex real part.
    unwrap      - Unwrap phase angle.
    isreal      - True for real array.
    cplxpair    - Sort numbers into complex conjugate pairs.

  Rounding and remainder.
    fix         - Round towards zero.
    floor       - Round towards minus infinity.
    ceil        - Round towards plus infinity.
    round       - Round towards nearest integer.
    mod         - Modulus (signed remainder after division).
    rem         - Remainder after division.
    sign        - Signum.
```

19. Evalúe la siguiente función en $x = 2{,}2$

$$f(x) = \frac{1}{0{,}1\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-2}{0{,}1}\right)^2}$$

```
>> f = (1/(0.1*sqrt(2*pi)))*exp(-0.5*(0.2/0.1)^2)
f =
    0.5399
```

20. Evalúe la función de variable independiente x y parámetros μ y σ

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

para $\mu = 2$ y $\sigma = 0{,}1$

```
>> mu = 2;
>> sigma = 0.1;
>> x = 2.2;
>> f = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma)^2)
f =
    0.5399
```

21. Evalúe la función anterior para valores de x que varíen entre 1 y 3 con incremento de 0.01

```
>> x = (1:0.1:3)';
>> f = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma).^2)
f =
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0013
    0.0443
    0.5399
    2.4197
    3.9894
    2.4197
    0.5399
    0.0443
```

```
    0.0013
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
```

3. Expresiones y Operadores

22. Dados los arreglos

$$A = \begin{bmatrix} -2 & 1 & 3 \\ 7 & 8 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 1 & -1 & 2 \\ -3 & 2 & 7 & 1 \\ 1 & 8 & 1 & -4 \end{bmatrix} \quad C = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

Efectuar las operaciones matriciales
a) AB b) $(AA')^{-1}$ c) $A\,(CC')^3\,B$

```
>> A = [-2 1 3; 7 8 1];
>> B = [0 1 -1 2; -3 2 7 1; 1 8 1 -4];
>> C = [2 1 3].';
>> A*B
ans =
     0    24    12   -15
   -23    31    50    18
>> (A*A.')^-1
ans =
    0.0718    0.0019
    0.0019    0.0088
>> A*(C*C.')^3*B
ans =
           0       32928        9408       -8232
           0      137200       39200      -34300
```

23. Dada las matrices

$$U = \begin{bmatrix} 2 & -1 \\ 3 & 0 \end{bmatrix} \quad V = \begin{bmatrix} 1 & -1 & 2 \\ 3 & 2 & 1 \end{bmatrix}$$

y utilizando las matriz A del ejemplo anterior, obtenga el resultado de las siguientes operaciones elemento a elemento (elementwise)
a) $A.*V$ b) $V./A$ c) $A.\hat{}V$ d) $V.*A+A$ e) $(AA')^U - U$

```
>> U = [2 -1; 3 0];
>> V = [1 -1 2; 3 2 1];
>> A
A =
    -2     1     3
     7     8     1

>> A.*V
ans =
    -2    -1     6
    21    16     1

>> V./A
ans =
   -0.5000   -1.0000    0.6667
    0.4286    0.2500    1.0000
```

```
>> A.^V
ans =
    -2     1     9
   343    64     1

>> V.*A+A
ans =
    -4     0     9
    28    24     2

>> (A*A.').^U - U
ans =
  194.0000    0.6667
  -30.0000    1.0000
```

■ **NOTA:** Cuando participan escalares en una expresión MATLAB, los operadores toman comportamientos particulares.
Sea el escalar k (arreglo de 1x1), entonces:

OPERACIÓN TIPO ARREGLO (elemento a elemento)		REGLA DE CORRESPONDENCIA
+/-	$B_{m,n} = k \pm A_{m,n}$	$b_{ij} = k \pm a_{ij}$
.*	$B_{m,n} = k\ .*\ A_{m,n} = A_{m,n}\ .*\ k$	$b_{ij} = k \cdot a_{ij}$
./	$B_{m,n} = k\ ./\ A_{m,n}$	$b_{ij} = k \cdot a_{ij}^{-1} = \dfrac{k}{a_{ij}}$
	$B_{m,n} = A_{m,n}\ ./\ k$	$b_{ij} = a_{ij} \cdot k^{-1} = \dfrac{a_{ij}}{k}$
.	$B_{m,n} = k\ .\backslash\ A_{m,n}$	$b_{ij} = k^{-1} \cdot a_{ij} = \dfrac{a_{ij}}{k}$
	$B_{m,n} = A_{m,n}\ .\backslash\ k$	$b_{ij} = a_{ij}^{-1} \cdot k = \dfrac{k}{a_{ij}}$
.^	$B_{m,n} = k\ .\hat{}\ A_{m,n}$	$b_{ij} = k^{a_{ij}}$
	$B_{m,n} = A_{m,n}\ .\hat{}\ k$	$b_{ij} = a_{ij}^{k}$

OPERACIÓN TIPO **MATRIZ** (elemento a elemento)		REGLA DE CORRESPONDENCIA
+/-	$B_{m,n} = k \pm A_{m,n}$	$b_{ij} = k \pm a_{ij}$
*****	$B_{m,n} = k * A_{m,n}$ $= A_{m,n} * k$	$b_{ij} = k \cdot a_{ij}$
/	$B_{m,n} = k \,/\, A_{m,n}$	$\nexists$
	$B_{m,n} = A_{m,n} \,/\, k$	$b_{ij} = a_{ij} \cdot k^{-1} = \dfrac{a_{ij}}{k}$
****	$B_{m,n} = k \setminus A_{m,n}$	$b_{ij} = k^{-1} \cdot a_{ij} = \dfrac{a_{ij}}{k}$
	$B_{m,n} = A_{m,n} \setminus k$	$\nexists$
^	$B_{m,n} = k \,\hat{}\, A_{m,n}$	$\exists$ solo si A es cuadrada y su cálculo utiliza los valores y vectores propios de A.
	$B_{m,m} = A_{m,m} \,\hat{}\, k$	$B_{m,m} = \underbrace{A_{m,m} \cdots A_{m,m}}_{k \text{ veces}}$

24. Dados los arreglos del ejemplo 21 y 22, obtenga el resultado de las siguientes operaciones matriciales con escalares

a) $A - 2V$ b) $U^5 - U^3$ c) $3AA' + 4U^{-2}$ d) $(BB')^{-2} + 2\left[\dfrac{A}{C'}\right]$ e) $\left[\frac{AV'UV}{(C+A'VC)'}\right]^{-1}$

```
>> A - 2*V
ans =
    -4     3    -1
     1     4    -1

>> U^5 - U^3
ans =
    -6    12
   -36    18

>> 3*A*A.' + 4*U^-2
ans =
   40.6667   -8.1111
  -11.6667  342.4444
```

```
>> (B*B.')^-2 + 2^[A;C.']
ans =
   51.3146   70.6763   38.9358
  351.7387  494.6379  243.7633
   73.6193  100.7770   58.4823

>> [A*V.'*U*V;(C+A.'*V*C).']^-1
ans =
   -0.2311    0.0053   -0.0152
    0.0971   -0.0036    0.0152
    0.1627   -0.0001    0.0152
```

25. Evalúe la función de variable independiente x y parámetros μ y σ

$$f\left(x;\mu,\sigma\right)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{\left(x-\mu\right)^{2}}{2\sigma^{2}}\right)$$

en $x\in[1{,}5;2{,}5]$ con paso $\sigma/10$, para $\sigma=0{,}1$, $\mu=2$

```
>> sigma=0.1;
>> mu=2;
>> x=(1.5:sigma/10:2.5)';
>> f = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma).^2)
f =
    0.0000
    0.0000
    0.0000
    0.0001
    0.0001
    0.0002
    0.0002
    0.0004
    0.0006
    0.0009
    0.0013
    0.0020
       :
    2.8969
    3.1225
    3.3322
    3.5207
    3.6827
    3.8139
    3.9104
    3.9695
    3.9894
    3.9695
    3.9104
    3.8139
    3.6827
    3.5207
    3.3322
    3.1225
    2.8969
       :
    0.0020
    0.0013
    0.0009
```

```
    0.0006
    0.0004
    0.0002
    0.0002
    0.0001
    0.0001
    0.0000
    0.0000
    0.0000
```

26. Construya un arreglo de tres columnas, en el que la primera esté constituida por el dominio, y las demás por el resultado de las evaluaciones de las funciones $f\left(x;\mu,\sigma\right)$ y $F\left(x;\mu,\sigma\right)$ sobre dicho dominio. Considere a $x \in [\mu - 3\sigma, \mu + 3\sigma]$ con paso $\Delta x_j = \sigma/10$, $\mu = 2$ y $F\left(x;u,\sigma\right)$ donde

$$F\left(x_i;u,\sigma\right) = \sum_{j=1}^{i} f\left(x_j;\mu,\sigma\right)\cdot\Delta x_j$$

```
>> mu = 2;
>> sigma = 0.1;
>> x = (1.5:sigma/10:2.5)';
>> f = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma).^2);
>> F = cumsum(f)*sigma/10;
>> R = [x f F]
R =
    1.5000    0.0000    0.0000
    1.5100    0.0000    0.0000
    1.5200    0.0000    0.0000
    1.5300    0.0001    0.0000
       :         :         :
    2.4700    0.0001    1.0000
    2.4800    0.0000    1.0000
    2.4900    0.0000    1.0000
    2.5000    0.0000    1.0000

>> whos x f F R
  Name        Size            Bytes  Class     Attributes

  F         101x1               808  double
  R         101x3              2424  double
  f         101x1               808  double
  x         101x1               808  double
```

27. Visualice en forma tabulada las funciones

$$\begin{aligned} x_{1,t} &= 1+\cos t \\ x_{2,t} &= 1-\sin t \end{aligned}$$

para $t \in [0, 2\pi]$ con paso $\pi/4$ en un único listado.

```
>> t = (0:pi/4:2*pi)';
>> x1 = 1+cos(t);
>> x2 = 1-sin(t);
>> U = [t x1 x2]
U =
         0    2.0000    1.0000
    0.7854    1.7071    0.2929
    1.5708    1.0000         0
    2.3562    0.2929    0.2929
```

```
    3.1416         0    1.0000
    3.9270    0.2929    1.7071
    4.7124    1.0000    2.0000
    5.4978    1.7071    1.7071
    6.2832    2.0000    1.0000
```

28. Determine la norma máxima y mínima del vector

$$\mathbf{v}_t = \begin{bmatrix} \cos 3t \\ \sin 2t \\ e^{\sin 4t} \end{bmatrix}$$

en $t \in [0, \pi]$. Utilice 1000 particiones del domino.

a) Con distribución lineal

b) Con distribución logarítmica

Finalmente active la visualiación en formato extendido (comando format , 15 dígitos) y compare los resultados

```
>> t1 = linspace(0,pi,1000)';
>> V1 = [cos(3*t1) sin(2*t1) exp(sin(4*t1))];
>> M1 = sqrt(sum(V1.^2)');
>> max1 = max(M1), min1 = min(M1)
max1 =
   47.7316
min1 =
   22.3495

>> t2 = logspace(0,pi,1000)';
>> V2 = [cos(3*t2) sin(2*t2) exp(sin(4*t2))];
>> M2 = sqrt(sum(V2.^2)');
>> max2 = max(M2), min2 = min(M2)
max2 =
   40.9275
min2 =
   21.0016

>> format long

>> max1, min1
max1 =
  47.731600822031240
min1 =
  22.349496638627013

>> max2, min2
max2 =
  40.927527362282810
min2 =
  21.001560109309661
```

29. Liste las funciones matriciales del álgebra lineal numérica (`help matfun`); luego, resuelva el sistema

$$\begin{aligned} 3x_1 + 4x_2 + x_3 + 5x_4 &= 2 \\ 2x_1 + x_2 + 9x_3 + 2x_4 &= 1 \\ x_1 + 2x_2 + x_3 + x_4 &= -2 \\ x_2 - x_4 &= 3 \end{aligned}$$

Efectuando todos los análisis previos antes de la obtención de la solución del sistema.

```
>> A = [3 4 1 5; 2 1 9 2; 1 2 1 1; 0 1 0 -1]
A =
     3     4     1     5
     2     1     9     2
     1     2     1     1
     0     1     0    -1

>> B = [2 1 -2 3]'
B =
     2
     1
    -2
     3

>> det(A)
ans =
    6.0000
>> inv(A)
ans =
    4.0000    1.0000  -13.0000    9.0000
   -1.1667   -0.3333    4.1667   -2.3333
   -0.5000    0.0000    1.5000   -1.0000
   -1.1667   -0.3333    4.1667   -3.3333

>> rank(A)
ans =
     4

>> rref(A)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>> [V,D] = eig(A)
V =
    0.6955    0.9172   -0.8344    0.1948
    0.6449   -0.2990    0.1083   -0.8581
    0.3080   -0.1299    0.0441    0.2695
    0.0742   -0.2290    0.5386    0.3913
D =
    7.6859         0         0         0
         0    0.3060         0         0
         0         0   -0.7990         0
         0         0         0   -3.1929

>> x = A\B
x =
   62.0000
  -18.0000
   -7.0000
  -21.0000
```

30. Dada la serie $\{x_t\}_{-10}^{10}$ donde

$$x_t = 20\frac{\sin t}{t}$$

liste los números enteros resultantes de las aproximaciones:
a) tendiendo a 0
b) tendiendo a $-\infty$
c) tendiendo a $+\infty$
d) tendiendo al entero mas cercano
Utilice una distribución lineal del tiempo en 20 observaciones.

```
>> t = linspace(-10,10,20)';
>> f = 20*sin(t)./t
f =
   -1.0880
    1.0271
    2.5312
    1.5500
       :
    1.0271
   -1.0880
>> a = [ fix(f) floor(f) ceil(f) round(f) ]
a =
    -1    -2    -1    -1
     1     1     2     1
     2     2     3     3
     1     1     2     2
     :     :     :     :
     1     1     2     1
    -1    -2    -1    -1
```

4. Indexación de matrices

31. Construya una matriz mágica de orden 7 y efectúe las siguientes operaciones

1. Obtenga en un arreglo P los elementos de A comprendidos entre las filas 2 y 5 y las columnas 1 y 4.

```
>> A = magic(7)
A =
    30    39    48     1    10    19    28
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
     5    14    16    25    34    36    45
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20
>> P = A(2:5,1:4)
P =
    38    47     7     9
    46     6     8    17
     5    14    16    25
    13    15    24    33
```

2. Obtenga en un arreglo Q las tres últimas columnas de A.

```
PRIMERA FORMA:
>> Q = A( 1:7 , 5:7 )
Q =
    10    19    28
    18    27    29
    26    35    37
    34    36    45
```

```
    42    44     4
    43     3    12
     2    11    20

SEGUNDA FORMA:
>> Q = A( : , 5:7 )
Q =
    10    19    28
    18    27    29
    26    35    37
    34    36    45
    42    44     4
    43     3    12
     2    11    20
```

3. Obtenga en un arreglo R las tres primeras filas de A.

```
>> R = A( 1:3 , : )
R =
    30    39    48     1    10    19    28
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
```

4. Crear un arreglo B que contenga las filas de A con las filas 1 y 4 intercambiadas.

```
>> B = A( [4 2 3 1 5 6 7], : )
B =
     5    14    16    25    34    36    45
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
    30    39    48     1    10    19    28
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20
```

5. Incrementar la fila 4 del arreglo B en 5 veces la fila 7

```
>> B(4,:) = B(4,:) + 5*B(7,:)
B =
     5    14    16    25    34    36    45
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
   140   194   248   246    20    74   128
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20
```

6. Asignar a las columnas 3 y 6 de A, las filas 2 y 4 del arreglo B respectivamente.

```
>> A(:,[3 6]) = B([2 4],:)'
A =
    30    39    38     1    10   140    28
    38    47    47     9    18   194    29
    46     6     7    17    26   248    37
     5    14     9    25    34   246    45
    13    15    18    33    42    20     4
    21    23    27    41    43    74    12
    22    31    29    49     2   128    20
```

7. Eliminar la fila 3 y la columna 5 del arreglo B.

```
>> B(3,:)=[], B(:,5)=[]
B =
     5    14    16    25    36    45
    38    47     7     9    18    27    29
   140   194   248   246    20    74   128
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20
```

8. Intercambiar las columnas 1 y 7 del arreglo A.

```
>> A = A(:,[7 2 3 4 5 6 1])
A =
    28    39    38     1    10   140    30
    29    47    47     9    18   194    38
    37     6     7    17    26   248    46
    45    14     9    25    34   246     5
     4    15    18    33    42    20    13
    12    23    27    41    43    74    21
    20    31    29    49     2   128    22
```

9. Listar los elementos del arreglo A como un único vector columna.

```
>> A(:)
ans =
    28
    29
    37
    45
     4
    12
    20
    39
    47
     6
    14
    15
    23
    31
     :
    30
    38
    46
     5
    13
    21
    22
```

10. Listar en un vector columna D los elementos de B que no exceden de la media de los elementos de B.

```
>> D = B( B < mean( B(:) ) )
D =
     5
    38
    13
    21
    22
```

```
14
47
15
23
31
16
 7
24
32
40
25
 9
33
41
49
36
27
44
 3
11
45
29
 4
12
20
```

11. Convertir en 0 los elementos de A que no excedan de la media de los elementos de A.

```
>> A( A < mean( A(:) ) ) = 0
A =
     0     0     0     0     0   140     0
     0    47    47     0     0   194     0
     0     0     0     0     0   248    46
    45     0     0     0     0   246     0
     0     0     0     0     0     0     0
     0     0     0     0    43    74     0
     0     0     0    49     0   128     0
```

Capítulo 3

El Lenguaje de Programación MATLAB

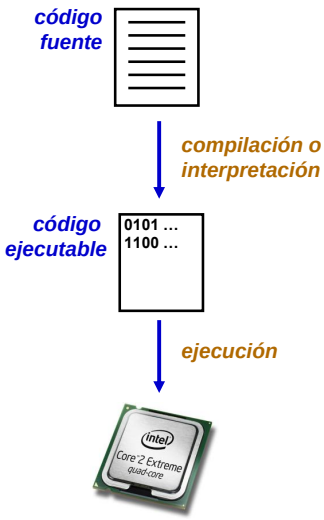
3.1. Los Programas

Un programa (también llamado programa informático o programa de computador) es simplemente un conjunto de instrucciones para una computadora escrita a través de un lenguaje de programación. Las instrucciones especificadas en un programa son ejecutadas por el procesador. Cuando se hace referencia a un programa se puede referir a un código fuente o a un código ejecutable

- Un **código fuente** (source code), es un archivo de texto que contiene instrucciones escritas en un determinado lenguaje de programación.
- Un c**ódigo ejecutable** (executable), es un archivo binario que contiene instrucciones que son de ejecución directa por el procesador.

De acuerdo a sus funciones, los programas pueden ser clasificados en

- Software de sistema; y
- Software de aplicación.



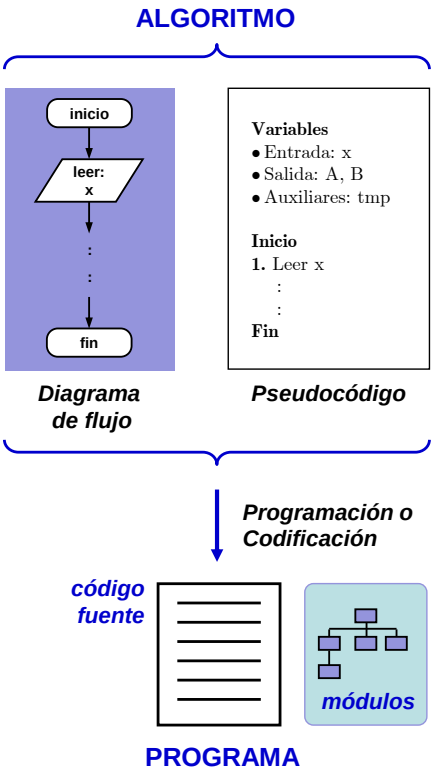
Observación: Un código ejecutable es el resultado de la compilación o interpretación a su equivalente en lenguaje máquina (ceros y unos) de cada una de las instrucciones especificadas en el código fuente.

3.2. Los Algoritmos y la Programación

Un algoritmo es una secuencia de pasos no ambigua, finita y ordenada que nos conduce a la solución de un problema. Se representan mediante Diagramas de Flujo o Pseudocógido. La programación es la

implementación (conversión) de un algoritmo, a través de un determinado lenguaje de programación, en un programa.

Los programas suelen subdividirse en partes menores (módulos), de modo que la complejidad algorítmica de cada una de las partes sea menor que la del programa completo, lo cual ayuda al desarrollo del programa.



Observación: Mientras que un algoritmo se ejecuta en una máquina abstracta que no tiene limitaciones de memoria o tiempo, un programa se ejecuta en una máquina real, que sí tiene esas limitaciones.

3.3. Los Lenguajes de Programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Un lenguaje de programación permite a uno o más programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser estos almacenados, transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural.

Una característica relevante de los lenguajes de programación es precisamente que más de un programador puedan tener un conjunto común de instrucciones que puedan ser comprendidas entre ellos para realizar la construcción del programa de forma colaborativa. Los procesadores usados en las computadoras son capaces de entender y actuar según lo indican programas escritos en un lenguaje fijo llamado lenguaje de máquina. Todo programa escrito en otro lenguaje puede ser ejecutado de dos maneras:

- Mediante un programa que va adaptando las instrucciones conforme son encontradas. A este proceso se le llama interpretar y a los programas que lo hacen se los conoce como intérpretes.
- Traduciendo este programa al programa equivalente escrito en lenguaje de máquina. A ese proceso se le llama compilar y al traductor se le conoce como compilador.

Observación: MATLAB posee un compilador que traduce las sentencias MATLAB en funciones equivalentes en lenguaje C; luego, se compila éste último para obtener así el código objeto a través de un compilador C para luego enlazarse con las bibliotecas matemáticas C de MATLAB junto a otros archivos que se disponga.

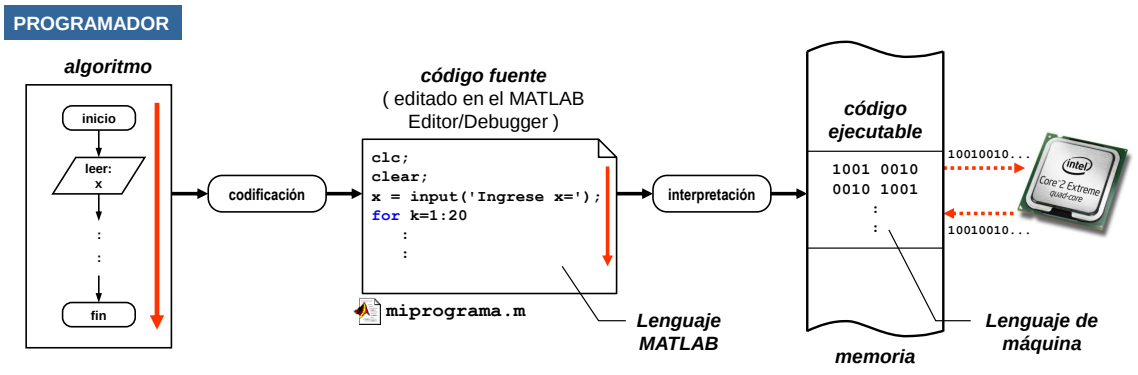
3.4. Clasificación de los Lenguajes de Programación

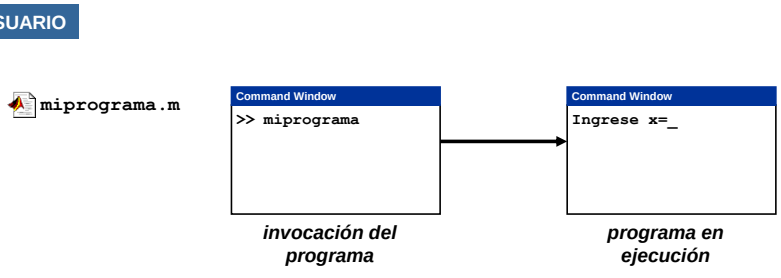
1. Por el nivel de abstracción
   a) **Lenguajes de bajo nivel:** Aquellos que mas se asemejan al lenguaje de una computadora (lenguaje de máquina)
   b) **Lenguajes de mediano nivel:** Aquellos conformados por nemónicos convertibles en forma directa a lenguaje máquina.
   c) **Lenguajes de alto nivel:** Aquellos que están conformados por elementos del lenguaje humano.
2. Por la forma de ejecución
   a) **Compilados:** Aquellos que convierten todo un programa a lenguaje máquina para su ejecución
   b) **Interpretados:** Aquellos que van convirtiendo sentencias de un programa a lenguaje máquina conforme vaya siendo necesario durante su ejecución (proceso de datos).
3. Por el paradigma de programación
   Un paradigma de programación es la filosofía utilizada en la construcción del software, podemos mencionar entre ellos a los paradigmas:
   a) Imperativo
   b) Funcional
   c) Lógico
   d) Orientado a Objetos
   e) Paralelo

El Lenguaje de Programación MATLAB es:

- Un Lenguaje de Programación de Alto Nivel
- Un Lenguaje de Programación Compilador e Interpretador
- Un Lenguaje de Programación Imperativo, Orientado a Objetos y Paralelo

3.5. Etapas de Ejecución de un Programa en MATLAB





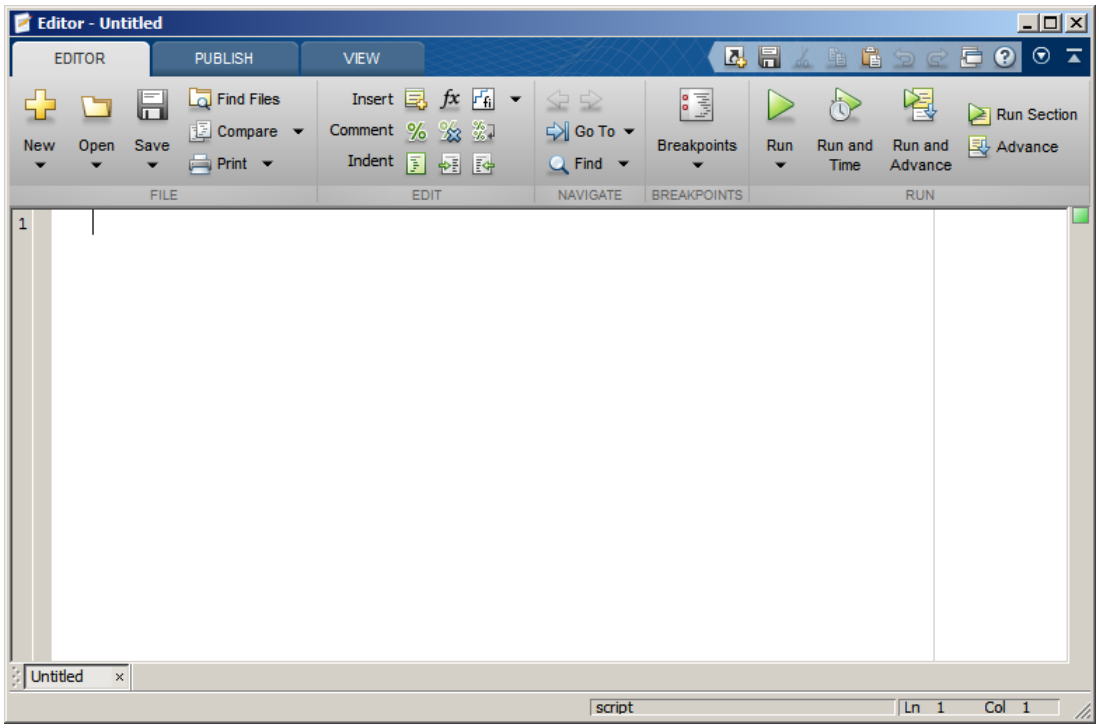
(se solicita la ejecución el programa)

3.6. Los Archivos M

Los archivos M (M-file, en inglés) son simples archivos de texto que contienen sentencias MATLAB. Es a través de ellos que se desarrolla la programación ó codificación. La extensión de éstos archivos es .m. El nombre de un archivo M es inmediatamente asociado al Sistema MATLAB como un nuevo comando. La edición/codificación de un programa en MATLAB se efectúa mediante la aplicación **MATLAB Editor/Debugger**.

Para iniciar el MATLAB Editor digitamos el comando edit desde la línea de comandos.

```
>> edit
```



Como se podrá observar, el editor creará por defecto un archivo M script vacío llamado Untitled, ubicando el cursor en la posición inicial listo para iniciar la codificación.

3.7. Tipos de Archivo M

Un archivo M puede ser de dos tipos:

- **Archivo M Script**
  - Contienen sentencias MATLAB.
  - En su llamada (invocación), no reciben ni retornan argumentos.
- **Archivo M Función**
  - Contienen sentencias MATLAB.
  - En su llamada(invocación), pueden recibir y retornan argumentos.

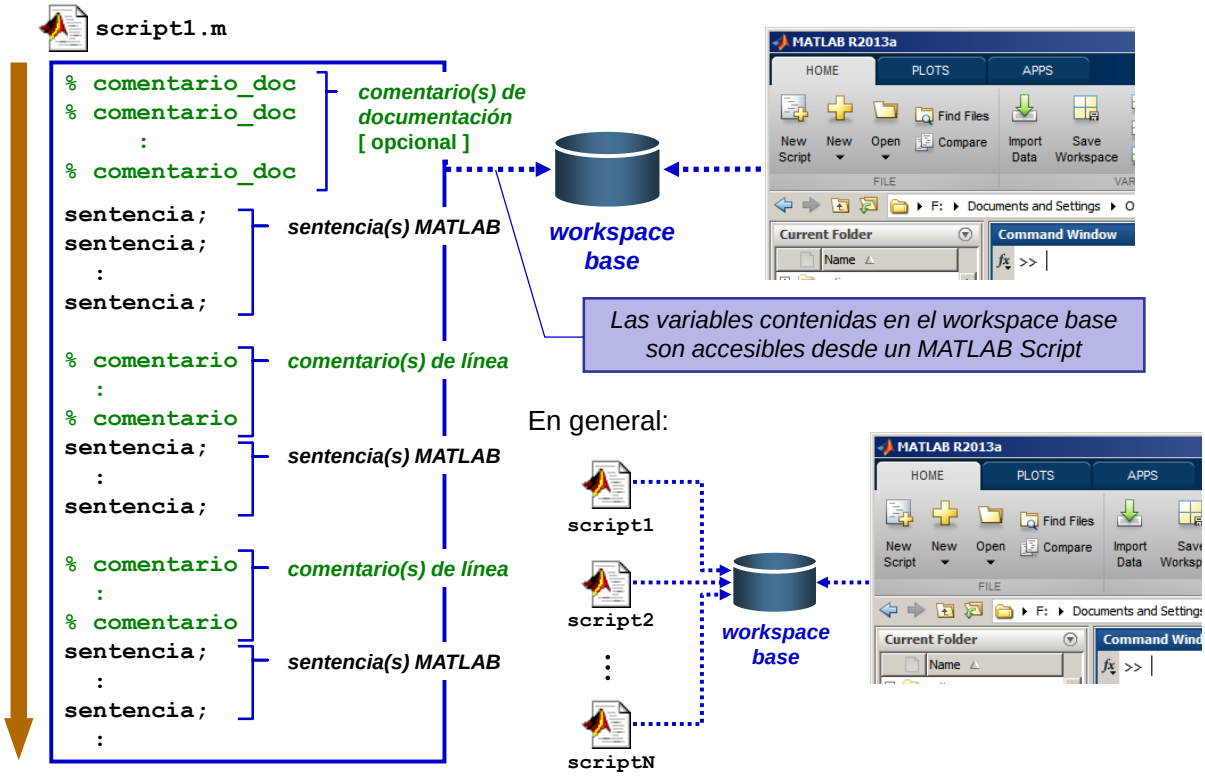
3.8. Los Archivos M – Script (MATLAB Scripts)

Se caracterizan por:

- Ser los archivos M mas simples.

- Son archivos externos que, generalmente, contienen secuencias de sentencias MATLAB, con la finalidad de automatizar bloques de comandos, tales como los utilizados en cálculo que requieran ser ejecutados repetidamente desde la línea de comandos u otro archivo M.
- Pueden operar con variables (datos) pre-existentes en el workspace base, o en su defecto crearlos y operar con ellos.
- Las variables creadas por los Scripts permanecen en el workspace base, siendo posible de ser reutilizadas en cálculos póstumos.
- No requieren la declaración de delimitadores de inicio/fin (begin/end).
- No retornan ni reciben argumentos.
- Pueden generar gráficos de salida usando comandos tales como plot.
- Pueden incluir líneas de comentario en cualquier posición, adjuntas a sentencias o como líneas de documentación del script.

3.9. Partes de un MATLAB Script



3.10. El comando input

Permite el ingreso de entradas del usuario.

- **Sintáxis:**
  - `variable_recepcion = input('mensaje')`
    Visualiza el texto mensaje como prompt en la pantalla, esperando la entrada numérica desde el teclado, y retorna el valor ingresado en variable_recepcion.
  - `variable_recepcion = input('mensaje', 's')`
    Visualiza el texto mensaje como prompt en la pantalla, esperando la entrada textual desde el teclado, y retorna el valor ingresado en variable_recepcion .

- **Observaciones**:
  - Si se presiona la tecla ENTER sin haber ingresado algo, input retorna una matriz vacía
  - Si se ingresa una entrada inválida, MATLAB mostrará el mensaje de error relevante y vuelve a mostrar el prompt solicitando una entrada válida.
  - Se puede especificar el caracter no imprimible nueva línea '\n'.
  - Para visualizar un backslash, use '\\'

3.11. El comando disp

Permite visualizar un texto o un arreglo.

- **Sintáxis**
  - `disp(X)`
    Muestra un arreglo, sin imprimir el nombre del arreglo. Si X contiene una cadena de texto, la cadena será mostrada.
- **Observaciones**
  - disp no visualiza arreglos vacíos.

3.12. El comando fprintf.

Permite escribir datos formateados en pantalla

- **Sintáxis**
  - `numBytes = fprintf( strFormato, var1, var2, ... )`
    Imprime en pantalla las variables var1,var2,... bajo el control de la cadena de formato strFormato y retorna el número de Bytes escritos en numBytes.
- **Cadena de Formato**
  - Permite controlar la notación, alineación, numero de dígitos significativos, ancho del campo, y otros aspectos de un formato de salida.
  - Puede también contener caracteres de escape que represente caracteres no imprimibles tales como nueva línea ('\n') o tabs ('\t')
  - Los especificadores de conversión inician con el caracter % seguido de los siguientes elementos: flag, ancho, precisión y carácter de conversión. (consulte tablas)

EJEMPLO: La especificación

```
%-12.7f
```

significa:

- **flag:** - (alineación a la izquierda)
- **ancho:** 12 (numero de espacios totales incluido el punto decimal)
- **precisión:** 7 (numero de dígitos decimales despues del punto)
- **carácter de conversión:** f (tipo de dato asociado al valor por imprimir, f es notación de punto fijo)

3.13. Los Archivos M – Función (MATLAB Function)

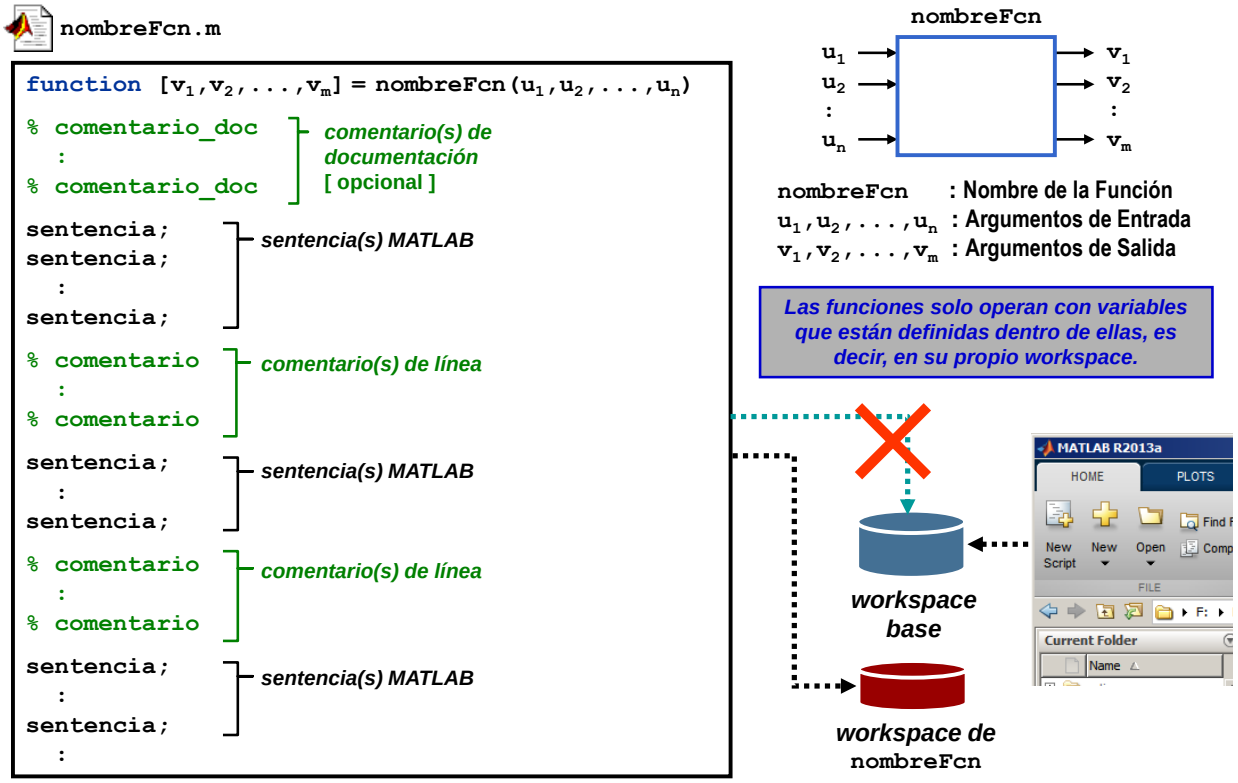
Son rutinas de programa, que pueden aceptar argumentos de entrada y retornar argumentos de salida.

`nombreFcn` : Nombre de la Función
$u_1, u_2, \ldots, u_n$: Argumentos de Entrada
$v_1, v_2, \ldots, v_m$: Argumentos de Salida

Cada función posee su propio workspace; el cual es independiente del workspace al que se accede desde el prompt de MATLAB. En otras palabras, las funciones solo operan con :

- Argumentos de Entrada.
- Variables que están definidas dentro de ellas.
- Variables globales (en caso sea necesario compartir variables entre diversos workspaceséstas deberán ser declaradas como globales en cada ámbito).
- Argumentos de Salida.

3.14. Partes de una función



3.15. Los Manipuladores de Función (function handle)

Un manipulador de función es un tipo de dato que contiene toda la información necesaria para la evaluación de una función. Son utilizados cuando se requiere que una función sea pasada como argumento de entrada a otra función. Se crean añadiendo el carácter @ antes del nombre de la función.

EJEMPLO: Crear un manipulador de la función sin de MATLAB y obtener el valor de $\sin(\pi/2)$ a través del manipulador

```
>> f1 = @sin
f1 =
   @sin

>> y = f1(pi/2)
y =
   1

>> whos f1
  Name  Size  Bytes  Class
  f1     1x1     16  function_handle
```

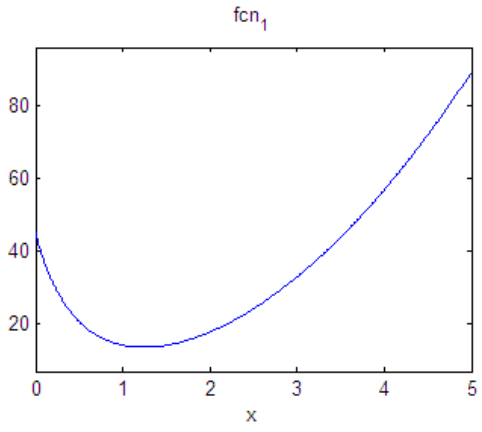
EJEMPLO: Crear un manipulador de función para la función matemática $f(x) = 3x^{2,1} + 45{,}3\,(x+1)^{-2}$

Primero, debemos crear un archivo M función que modele la función matemática

```
1 function y = fcn1(x)
2 y = 3*x.^2.1 + 45.3*(x+1).^-2;
```

Luego, creamos desde la línea de comando (o desde algún script) un manipulador a la función fcn1 recien creada, posteriormente podremos utilizar este manipulador para evaluar la función fcn1.

```
1 >> f2 = @fcn1
2 f2 =
3    @fcn1
4
5 >> y = fcn1(0)
6 y =
7    45.3000
8
9 >> ezplot(@fcn1,[0 5])
```



3.16. Las Funciones Anónimas

Son un medio de proveer la creación de funciones en línea sin la escritura de un archivo M. La función anónima es referenciada a través de un manipulador de función.

- **Sintaxis:**

```
f = @(arg1, arg2, ...) regla_de_correspondencia
```

EJEMPLO: Implementar mediante funciones anónimas las siguientes funciones matemáticas

$$f_1(x,y) = xe^{-x^2-y^2} \quad f_2(x,y,z) = f_1(x,y)\sqrt{z+2}$$

y obtener los siguientes valores $f_1(2,0{,}5)$ y $f_2(2,1,10)$

```
>> f1 = @(x,y) x.*exp(-x.^2-y.^2);
>> f1(2,0.5)
ans =
    0.0285

>> f2 = @(x,y,z) f1(x,y)*sqrt(z+2);
>> f2(2,1,10)
ans =
    0.0467
```

EJEMPLO: Implemente el algoritmo de integración por el método del trapecio de manera que la función por integrar sea un argumento de entrada del tipo manipulador de función

■ `trapecio.m`

```
function I = trapecio(fhandle, a, b, N)
% TRAPECIO calcula la integral por el método del trapecio
%
% Entradas:
% - fhandle: función a integrar
% - a : límite inferior
% - b : límite superior
% - N : número de particiones
%
% Salida:
% - I : Integral aproximada

h = (b-a)/N;
S = sum(feval(fhandle,a+(1:N-1)*h));
I = (feval(fhandle,a) + 2*S + feval(fhandle,b))*h/2;
```

EJEMPLO: Calcule las integrales

$$I = \int_0^{\pi} \sin(x)dx$$

implementando las funciones a integrar mediante funciones anónimas.

```
>> f =@(x) sin(x);
>> I = trapecio(f,0,pi,1000)
I =
   2.0000
```

3.17. Las Subfunciones

Una función implementada a través de un archivo M puede contener otras funciones, denominadas subfunciones, las cuales aparecen a continuación de la función primaria (principal). Las subfunciones son visibles solo por la función principal y cualquier otra subfunción.

■ `funcionprincipal.m`

```
function [ ... ] = funcionprincipal(...)
% documentacion de funcionprincipal
% :
...
```

```
...
function [ ... ] = subfuncion1(...)
% documentacion de subfuncion1
% :
...
...
function [ ... ] = subfuncion2(...)
% documentacion de subfuncion2
% :
... ...
```

EJEMPLO: Analice el siguiente codigo fuente

■ `newstats.m`

```
function [avg, med] = newstats(u) % Función Primaria
% NEWSTATS Encuentra la media y la medianta
n = length(u);
avg = mean(u, n);
med = median(u, n);

function a = mean(v, n)    % Subfunción
% Calcula el promedio.
a = sum(v)/n;

function m = median(v, n) % Subfunción
% Calcula la mediana.
w = sort(v);
if rem(n, 2) == 1
        m = w((n+1) / 2);
else
        m = (w(n/2) + w(n/2+1)) / 2;
end
```

3.18. Visibilidad y alcance de las variables

Las variables creadas en la ventana de comandos o en un script residen en un área de memoria denominada workspace base. Toda función posee su propia área de memoria asignada, su propio workspace, en la que residen sus argumentos de entrada, de salida y los creados dentro de la función.

- **Variables Locales**

  Por defecto, las variables del workspace de una función son solo accesibles desde la misma función, por lo que se acostumbran llamarlas variables locales.

- **Variables Globales**

  Las variables que se requieran compartir entre los contextos:

  - Dos o mas funciones
  - Un script y una o más funciones
  - La ventana de comandos y una función

  Se denominan variables globales (en su contexto) y deben ser declaradas como tales en cada uno de los espacios (script, función o ventana de comando) donde se desee ser referenciada.

```
global var1, var2, ... ;
```

- **Variables Persistentes**

  Las variables locales a una función cuyos valores son retenidos en memoria, entre llamadas a la función, se denominan variables persistentes. Éstas variables son eliminadas de memoria cuando se modifica o limpia (clear) la función.

```
persistent var1, var2, ... ;
```

Capítulo 4

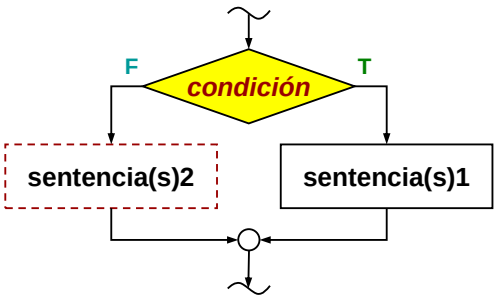
Diseño e implementación de algoritmos numéricos

4.1. Sentencias de Control Selectivas

4.1.1. Sentencias de Control Selectivas Simple

Por evaluación de condición: if ... else

- **Diagrama de Flujo**



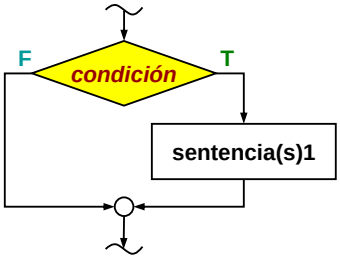
- **Sintáxis**

```
if condición
  sentencia(s)1
else
  sentencia(s)2
end
```

donde:

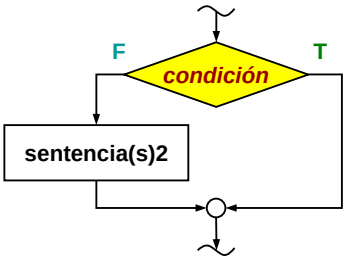
- condición, es la expresión (lógica orelacional) a evaluarse. Su resultado es deltipo logical (1= true, 0= false).
- sentencia(s)1, puede ser una o massentencias a ejecutarse siempre quecondición = true.
- sentencia(s)2, puede ser una o massentencias a ejecutarse siempre quecondición = false. (es opcional).

- **Casos Especiales**
  - Ausencia de sentencia(s)2




```
if condición
  sentencia(s)1
end
```

  - Ausencia de sentencia(s)1

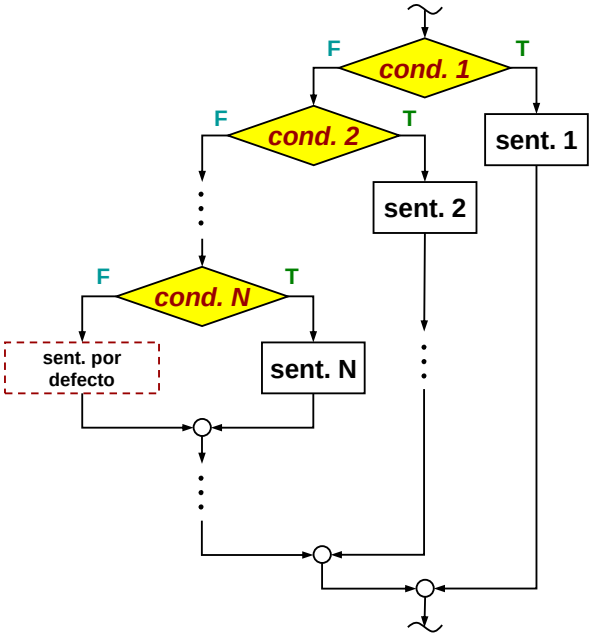



```
if condición
  ;
else
  sentencia(s)2
end
```

4.1.2. Sentencias de Control Selectivas Múltiple

Por consecutivas evaluaciones de condiciones : if ... elseif ... else

- **Diagrama de Flujo**



- **Sintáxis**

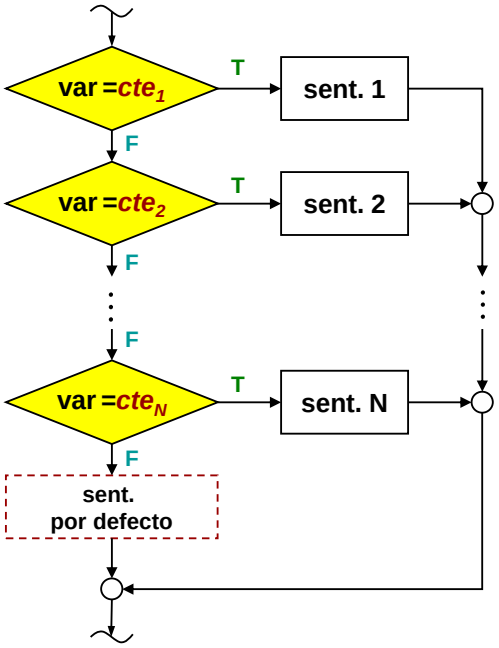
```
if cond. 1
  sent. 1
elseif cond. 2
  sent. 2
     :
elseif cond. N
  sent. N
else
  sent. por defecto
end
```

donde:

- cond.k, es la expresión k-ésima (lógica o relacional) a evaluarse. Su resultado es del tipo logical (1=true, 0=false).
- sent.k, es la sentencia(s) a ejecutarse siempre que cond.k =true.
- sent. por defecto, es la sentencia(s) a ejecutarse por defecto, o sea, cuando cond.1=cond.2=...=cond.N=false (es opcional).

Por múltiples comparaciones: switch ... case ... otherwise

- **Diagrama de Flujo**



- **Sintáxis**

```
switch var
  case CTE_1
     sent. 1
  case CTE_2
     sent. 2
        :
  case CTE_N
     sent. N
  otherwise
     sent. por defecto
end
```

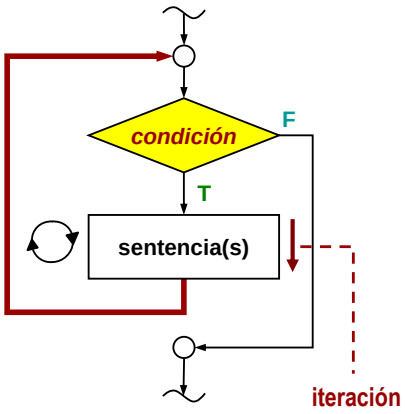
donde:

- var, es una variable entera o caracter.
- CTE_k, es la constante k-ésima de comparación.
- sent.k, es la sentencia(s) k-ésima a ejecutarse cuando la comparación var =ctek arroje como resultado true.
- sent. por defecto, es la sentencia(s). a ejecutarse cuando todas las comparaciones var =ctek arrojen como resultado false

4.2. Sentencias de Control Iterativa

4.2.1. Por evaluación de condición: while

- **Diagrama de Flujo**



- **Sintáxis**

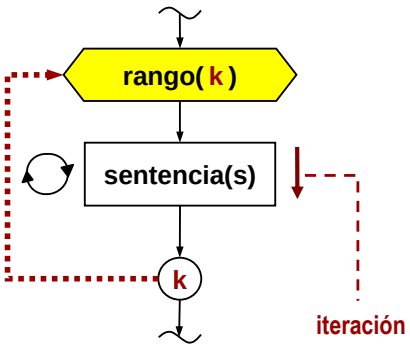
```
while condición
  sentencia(s)
end
```

donde:

- condición, es la expresión (lógica o relacional) a evaluarse. Su resultado es del tipo logical (1= true, 0= false).
- sentencia(s), es la sentencia(s) a ejecutarse siempre que la evaluación de la condición arroje como resultado el valor de true

4.2.2. Por recorrido de contador: for

- **Diagrama de Flujo**



- **Sintáxis**

```
for rango(k)
  sentencia(s)
end
```

donde:

- rango(k), es el rango de valores que toma la variable contadora : k. Por ejemplo, considere como rango(k) a

  `k=1:3:7`

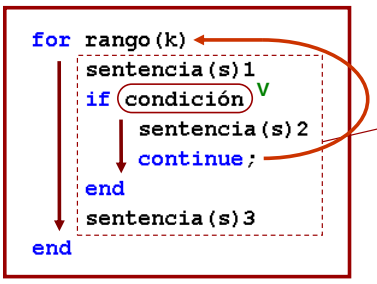
  entonces k tomara los valores 1, 4 y 7.
- sentencia(s), es la sentencia(s) a ejecutarse para cada uno de los valores del contador. ej: Para el caso anterior, se ejecutará la sentencia(s) 3 veces: la primera para k=1, la segunda para k=4 y la tercera para k=7.

4.3. Sentencias Especiales

4.3.1. Sentencia de salto: `continue`

Pasa el control a la siguiente iteración en los bucles for o while en el cual aparezca, salteando al posible conjunto de sentencias del cuerpo del bucle que la sucedan

EJEMPLO: Analizar

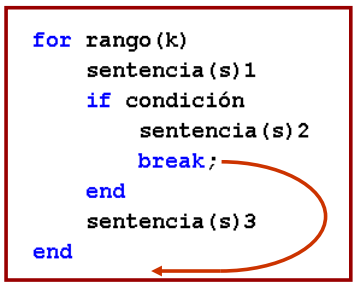


- `rango(k)` define los valores de `k` para cada uno de los cuales se efectuará una iteración
- El cuerpo del bucle for contiene una sentencia condicional `if` que evalúa una `condición`.
- La sentencia de salto `continue` se ejecutará siempre que la evaluación de `condición` resulte verdadera.
- Al ejecutarse `continue` se iniciará una nueva iteración, salteando las instrucciones que preceden a `continue` ; es decir `sentencia(s)3` .

4.3.2. Sentencia de ruptura: `break`

Termina la ejecución de un bucle for o while. Las sentencias que aparezcan después de la sentencia break, no serán ejecutadas.

EJEMPLO: Analizar



- La sentencia de salto `break` se ejecutará siempre que la evaluación de `condición` resulte verdadera.
- Al ejecutarse `break` se finalizará la ejecución del bucle for obviando la ejecución de las sentencias posteriores; es decir `sentencia(s)3` .

4.3.3. Sentencia de terminación: `return`

Ocasiona un normal retorno a la función invocante.

EJEMPLO: Analizar

```
function d = det(A)
%DET det(A) es el determinante de A.
if isempty(A)
    d = 1;
    return
else
    ...
end
```

Observación: isempty(A) es una función que retorna true siempre que la matriz A sea vacía: [ ]

```
>> A = [3 4; 8 11];
>> detA = det(A)
detA =
     -29
>> B = [];
>> detB = det(B)
detB =
     1
>>
```

4.4. Introducción a los Métodos Numéricos

4.4.1. Los Métodos Numéricos

Los Métodos Numéricos (Análisis Numérico) son la rama de las matemáticas que se encargan de diseñar algoritmos para, a través de números y reglas matemáticas simples, simular procesos matemáticos más complejos aplicados a procesos del mundo real; es decir, resolver el modelo que los explica. Entre los mas aplicados a economía computacional podemos mencionar:

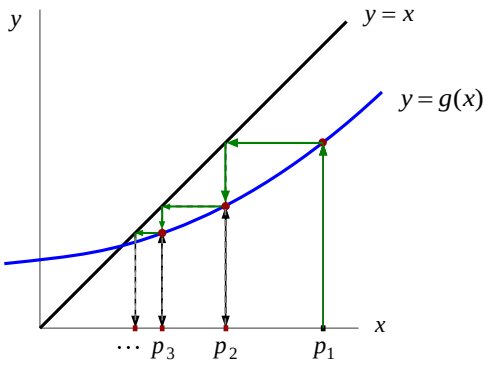
- Métodos para le Resolución de ecuaciones no lineales.
- Métodos para la Resolución de sistemas lineales.
- Métodos para la Interpolación y aproximación polinomial.
- Métodos para el Ajuste de curvas.
- Métodos para la Derivación numérica.
- Métodos para la Integración numérica.
- Métodos para la Optimización numérica.
- Métodos para la Resolución de Ecuaciones diferenciales.
- Métodos para el Cálculo de Valores y Vectores Propios.

La mayoría de softwares en la economía computacional, traen implementados los métodos numéricos, a través de bibliotecas; mientras que otros permiten la adaptabilidad de los mismos según el caso en análisis.

4.4.2. Solución de Ecuaciones No Lineales

El Método del Punto Fijo

Aproxima la solución de la ecuación $x = g(x)$ iniciando con el valor inicial de partida p_1 y la fórmula de recurrencia $p_n = g(p_{n-1})$



VALOR INICIAL → p_1

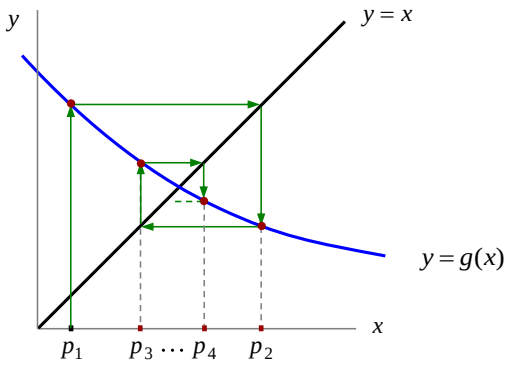
$p_2 = g(p_1)$

$p_3 = g(p_2)$

$\vdots$

$p_n = g(p_{n-1})$

EL METODO CONVERGE



VALOR INICIAL → p_1

$p_2 = g(p_1)$

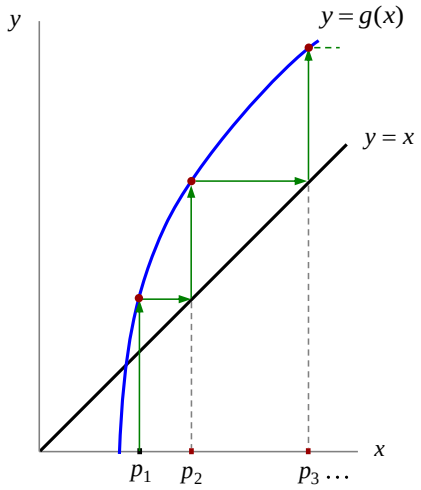
$p_3 = g(p_2)$

$p_4 = g(p_3)$

$\vdots$

$p_n = g(p_{n-1})$

EL METODO CONVERGE



VALOR INICIAL → p_1

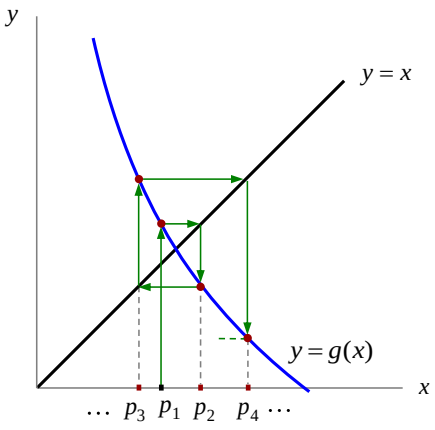
$p_2 = g(p_1)$

$p_3 = g(p_2)$

$\vdots$

$p_n = g(p_{n-1})$

EL METODO DIVERGE



VALOR INICIAL → p_1

$p_2 = g(p_1)$

$p_3 = g(p_2)$

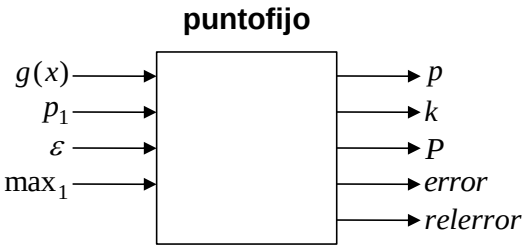
$p_4 = g(p_3)$

$\vdots$

$p_n = g(p_{n-1})$

EL METODO DIVERGE

- **Argumentos de Entrada/Salida**



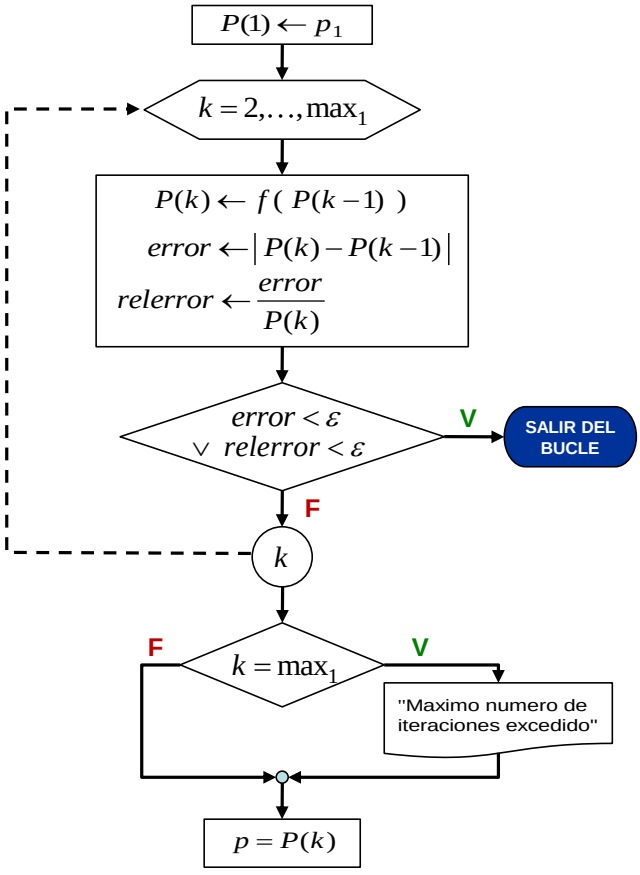
Entrada:

- $g(x)$:función iterativa
- p_1: valor inicial para el método de punto fijo
- ε: tolerancia
- max_1: máximo número de iteraciones

Salida:

- p: aproximación resultante por el método fijo
- k: numero de iteraciones efectuadas
- P: vector columna conteniene la secuencia de aproximaciones $\{P(1), P(2), \ldots, P(k)\}$
- $error$: error cometido en la aproximación p

- **Algoritmo (Diagrama de Flujo)**



- **Codificación**

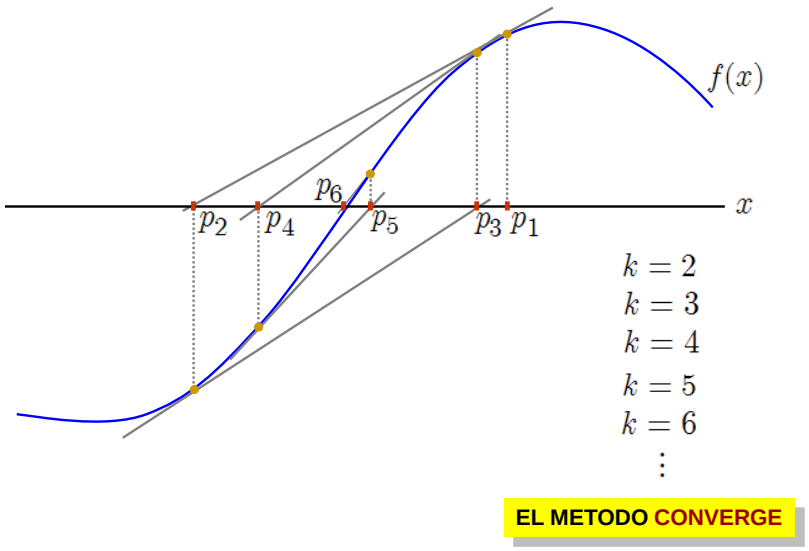
```
function [p,k,P,err] = puntofijo(strg,p1,tol,max1)
% ENTRADA:
% strg - es la función iterativa ingresada como cadena
%   p1 - es el valor inicial para el metodo de punto fijo
%  tol - es la tolerancia
% max1 - es el máximo numero de iteraciones
% SALIDA:
%    p - es la aproximación por el método de punto fijo
%    k - es el numero de iteraciones efectuadas
%    P - contiene la secuencia {pn}
%  err - es el error en la aproximación
P(1) = p1;
for k=2:max1
    P(k) = feval(strg,P(k-1));
    err = abs( P(k)-P(k-1) );
    relerr = err / ( P(k)+eps );
    if ( err<tol | relerr<tol)
        break;
    end
end

if k==max1
    disp('Numero máximo de iteraciones excedido');
end

P = P(:);
p = P(end);
```

El Método de Newton-Raphson

Permite aproximarnos a una raíz de a partir de $f(x)$ un valor inicial p_1 mediante la fórmula de recurrencia (iteración)

$$p_k = p_{k-1} - \frac{f(p_{k-1})}{f'(p_{k-1})}$$

para $k = 2, 3, \ldots$



- **Codificación**

```
function [p,P,error,k] = newton(strf,strdf,p1,tol,kmax)
P(1) = p1;
for k=2:kmax
    P(k) = P(k-1)-feval(strf,P(k-1))/feval(strdf,P(k-1));
    error(k) = abs(P(k)-P(k-1));
    if error(k)<tol
        break;
    end
end
if k == kmax
    disp('Numero máximo de iteraciones excedido');
end
p = P(end);
P = P.';
error = error.';
```

EJEMPLO: Resolver $f(x) = x - e^{-x} = 0$ con una precisión de 10^{-4}. Use a lo mas 20 iteraciones.

Primero preparamos las funciones MATLAB para $f(x) = x - e^{-x}$ y $f'(x) = 1 + e^{-x}$

■ `fcn1.m`

```
function y=fcn1(x)
y = x-exp(-x);
```

■ `fcn2.m`

```
function y = fcn2(x)
y = 1+exp(-x);
```

Luego, invocamos a la función newton con los parámetros necesarios

```
>> [p,P,error,k] = newton('fcn1','fcn2',1.2,1e-4,20)
p =
    0.5671
P =
    1.2000
    0.5092
    0.5665
    0.5671
    0.5671
error =
         0
    0.6908
    0.0573
    0.0006
    0.0000
k =
     5
>>
```

raíz aproximada

Sucesión de aproximaciones

Sucesión de errores por cada aproximación

Número de iteraciones utilizadas

EJERCICIOS

Sentencias Condicionales

I. Simple

`if`

1. Implemente un script que permita verificar si un numero es par
2. A partir del script anterior, crear una versión en forma de función MATLAB.

`if ... else`

1. Implemente una función que permita modelar la función matemática valor absoluto $y = |x|$
2. Implemente una función que permita modelar la función matemática signo $y =$sgn(x)
3. Implemente una función que determine las raíces de una ecuación cuadrática.
4. Implemente una función que permita verificar que un numero escalar x se encuentra dentro de un intervalo $[a, b]$ dado
5. Implemente una función que permita verificar si cada uno de los elementos de una matriz $\mathbf{X}$ se encuentra dentro de un intervalo $[a, b]$ dado
6. Implemente una función que permita verificar el cumplimiento de una hipótesis dado el valor del t-estadístico y el nivel de significancia α de una prueba de dos colas.

CASO: `if ... else` anidados

1. Implemente una función que permita verificar si un año es bisiesto.

   Un año es bisiesto si es divisible por 4, excepto el último de cada siglo (aquel divisible por 100) salvo que este último sea también divisible por 400.
2. Implemente una función que clasifique a un triángulo según la longitud de sus lados: “Equilatero”, “Escaleno” o “Isósceles”.
3. Haga las modificaciones a la función anterior para el caso en el que el triángulo no exista, esto es, la longitud de los lados no formen un triángulo.
4. Implemente una función que determine el mayor de tres numeros.

II. Múltiple

`if ... elseif ... else`

1. Implementar una función que permita modelar la siguiente función compuestas

$$y = \begin{cases} -1 & , \ x < -1 \\ 2x + 1 & , \ -1 \leq x < 0 \\ e^{-x} & , \ x \geq 0 \end{cases}$$

2. Implemente una función que permita identificar el intervalo al que pertenece una variable x según la siguiente tabla

Intervalo	x
I	< 5
II	$[5, 10\rangle$
III	$[10, 15\rangle$
IV	≥ 15

`switch ... case`

1. Implementar un menu de opciones que permita realizar las operaciones aritméticas básicas: suma, resta, multiplicación y división de dos numeros
2. Ingresar un numero entero, y si este termina en 2,5 u 8 reportar el cuadrado del numero, si este termina en 4,7 o 9 reportar el numero multiplicado por 5 y reportar el mismo número en otro caso.
3. Ingresar el numero de mes y el año y reporte el número de días que tiene ese mes.
4. Dados como entrada 3 enteros representando la fecha como día, mes, año, imprimir la fecha del día anterior. Por ejemplo para una entrada como: 1/3/1992 La salida será: Fecha anterior a 1/3/1992 es 29/02/1992.

Sentencias Repetitivas

I. Controlada por expresiones

`while`

1. Implementar una función que tome como entrada un $n-$vector $\mathbf{x}$ y retorne el $n-$vector y tal que

$$y_j = \sum_{i=1}^{j} x_i$$

   para todo $j = 1, 2, ..., n$
2. Implementar una función que retorne el MCD de dos números enteros.
3. Implementar una función que retorne en un vector columna los numeros primos comprendidos entre 1 y n. El valor n deberá ser pasado a la función como argumento de entrada.

   Sugerencia: Use el algoritmo de la Criba de Eratóstenes
4. Implementar un programa que haga la lectura de une determinada fecha dd/mm/aa y determine el numero de dias transcurridos desde el inicio de tal año.
5. Implementar una función que efectúe la lectura de un número de tal manera que éste pertenezca al intervalo $[0, 10]$

II. Controlada por conteo

`for`

1. Implementar una función que permita evaluar la sumatoria

$$S = \sum_{k=1}^{n} \frac{x^k}{k!}$$

2. Implementar una función que permita calcular el factorial de un número.
3. Implementar una función que determine los N primeros elementos de la *serie de fibonacci*: $0, 1, 1, 2, 3, 5, ...$
4. Implementar una función que mediante el proceso de división sintética permita evaluar un polinomio.
5. Implementar una función que permita determinar si un número es perfecto. Se dice que un número es perfecto cuando la suma de sus divisores (menor que el numero) es igual a el mismo.
6. Implementar una función que permita conocer los números perfectos comprendidos entre 1 y N. ($N \geq 100$).
7. Implementar una función que obtenga las raíces de una función no lineal por el *método de Newton-Raphson*.

8. Implementar una función que obtenga las raíces de un sistema de funciones no lineal por el *método de Newton-Raphson*.

9. Implementar una función que permita construir la *matriz de Hilbert*.

10. Implementar una función que permita construir la *matriz de Pascal*.

11. Implementar una función que permita construir una *matriz de identidad*.

12. Implementar una función que permita construir una *matriz mágica*. Considere el caso impar.

13. Implementar una función que ordene los elementos de un vector columna de menor a mayor por el *método burbuja*.

14. Implemente una aplicación que emule el funcionamiento de un cajero automático. El cajero solo despacha billetes en denominacion nacional de S/.10, S/.20, S/.50, S/.100 y S/.200. Considere que el cajero inicia sus operaciones con un numero específico de cada una de las denominaciones y que conforme los usuarios van haciendo retiros, las provisiones de cada denominación van disminuyendo. En caso ya no existan denominaciones para despachar un pedido del cliente se le sugerirá un monto que pueda ser despachado lo mas cercano al monto que deseaba retirar del cajero.

15. n participantes se disponen formando un circulo, asignándose a cada participante un numero diferente entre 1 y n. Se inicia un conteo m posiciones a partir del participante No. 1. El participante en el que termina el conteo es retirado del circulo, los participantes cierran filas y se procede a reiniciar el conteo a partir del participante inmediato al que fue retirado. El juego finaliza cuando han sido retirado $n-1$ participantes, siendo el último el participante elegido (sacrificado).

    a) Implementar un programa que indique la secuencia en la que los participantes van siendo retirados del círculo así como el participante que resulta ser sacrificado al finalizar todos los conteos.

    b) Modifique el programa anterior de manera que el participante a partir del cual se inicia el juego sea especificado por el usuario.

    c) Modifique el programa anterior de manera que el paso del conteo (m) sea elegido aleatoreamente con un par de dados cada vez que se inicie un conteo.

    d) Imagine que existe un jugador que se desea innmolar por sus demas compañeros y desea ser el elegido. Modifique el programa de manera que para un numero n determinado de jugadores, le proporcione el paso m adecuado. Considere m como un valor fijo que se especifica al inicio de todo el juego mortal.

Funciones Recursivas

1. Implementar una función recursiva para la obtención de la *Serie de Fibonacci*

2. Implementar una función recursiva para la obtención del *factorial de un numero*

3. Implementar la *función de Ackermann*.

    La *función de Ackermann* es una función recursiva que toma dos números naturales como argumentos y devuelve un único número natural. Como norma general se define como sigue:

$$A(m,n) = \begin{cases} n+1 & , \text{ si } m = 0 \\ A(m-1,1) & , \text{ si } m > 0 \text{ y } n = 0 \\ A(m-1, A(m,n-1)) & , \text{ si } m > 0 \text{ y } n > 0 \end{cases}$$

4. Implementar un programa en MATLAB que resuelva el problema "*Las torres de Hanoi*".

    Se tienen 3 palos de madera, que llamaremos palo izquierdo, central y derecho. El palo izquierdo tiene ensartados un montón de discos concéntricos de tamaño decreciente, de manera que el disco mayor está abajo y el menor arriba.

    El problema consiste en mover los discos del palo izquierdo al derecho respetando las siguientes reglas:

    a) Sólo se puede mover un disco cada vez.

b) No se puede poner un disco encima de otro más pequeño.

c) Después de un movimiento todos los discos han de estar en alguno de los tres palos.

Implementar una función que tome cono entrada un valor n, y retorne la secuencia de pasos para resolver el problema.

Capítulo 5

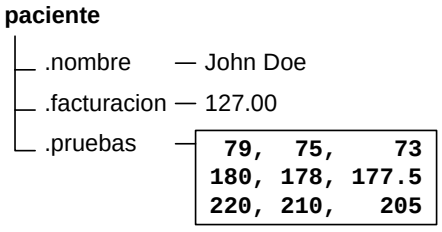
Estructuras de datos avanzadas

5.1. Tipos de Datos Avanzados

5.1.1. Estructuras

Una *estructura* (*struct*) es un tipo de dato MATLAB que agrupa datos relacionados usando contenedores denominados *campos* (fields). Los datos contenidos en cada campo pueden ser de cualquier tipo o tamaño.

EJEMPLO: A continuación se presenta una estructura que permite almacenar el registro de un paciente cuyos campos son nombre (arreglo fila del tipo char), facturación (escalar del tipo double) y pruebas (arreglo matricial del tipo double).




```
>> paciente.nombre = 'John Doe';
>> paciente.facturacion = 127.00;
>> paciente.pruebas = [79, 75, 73; 180, 178, 177.5; 220, 210, 205];
>> paciente
paciente =
      nombre: 'John Doe'
 facturacion: 127
     pruebas: [3x3 double]
```

Como se puede apreciar, la construcción de la estructura consiste en la creación de un primer registro. Se asigna al campo nombre del la estructura paciente la cadena 'John Doe', luego se asigna al campo facturación de la estructura paciente el valor 127.00, y finalmente se asigna al campo pruebas de la estructura paciente la matriz $\begin{pmatrix} 79 & 75 & 73 \\ 180 & 178 & 177{,}5 \\ 220 & 210 & 205 \end{pmatrix}$.

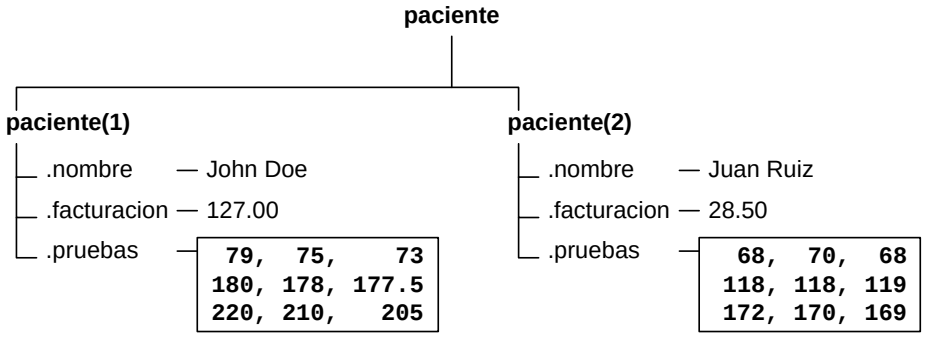
5.1.2. Arreglo de estructuras

Un arreglo de estructuras se crea añadiendo mas estructuras a una previamente definida indexando como si se tratase de un arreglo.

EJEMPLO: Añadir un segundo registro a la estructura paciente, donde el nombre es 'Juan Ruiz', su facturación es 28.5 y la prueba que presenta es $\begin{pmatrix} 68 & 70 & 68 \\ 118 & 118 & 119 \\ 172 & 170 & 169 \end{pmatrix}$

Para lograr esto, bastará con ingresar los campos del nuevo paciente asignándole una posición distinta a la primera en el arreglo de estructuras, en este caso usaremos la segunda posición, y se procede a asignar los valores a cada uno de los campos respectivamente.

```
>> paciente(2).nombre = 'Juan Ruiz';
>> paciente(2).facturacion = 28.50;
>> paciente(2).pruebas = [68, 70, 68; 118, 118, 119; 172, 170, 169];
>> paciente
paciente =
1x2 struct array with fields:
    name
    billing
    test
```



Observe que la forma por defecto que adopta el arreglo de estructuras es el de una fila.

Propiedades de los arreglos de estructuras

- Todas las estructuras en el arreglo tienen el *mismo número de campos*.
- Todas las estructuras tienen los *mismos nombres de campo*.
- Los campos del mismo nombre en diferentes estructuras pueden contener diferentes tipos o tamaños de dato.
- Cualquier campo no especificado para una nueva estructura en el arreglo contiene arreglos vacios (`[ ]`).
- El acceso a los datos se puede hacer haciendo uso de la *notación punto* de la forma

  `NombreEstructura.NombreCampo`

- Para acceder a parte de un campo, se debe añadir índices apropiados según el tamaño y el tipo de datos en el campo.

5.1.3. Arreglo Celda (Cell Arrays)

Un arreglo celda es un tipo de dato con contenedores de datos indexados llamados *celdas*. Cada celda contiene *cualquier tipo de dato*. Los arreglos celda comúnmente contienen listas de cadenas de texto, combinaciones de texto y numeros a partir de hojas de cálculo o archivos de texto, o arreglos numéricos de diferentes dimensiones (tamaños).

Hay dos formas de referenciar a los elementos de un arreglo celda:

1. Encerrando los indices en *paréntesis*, `( )`, para referenciar a conjuntos de celdas — por ejemplo, para definir un subconjunto del arreglo.
2. Encerrando los indices en *llaves*, `{ }`, para referenciar texo, numeros o cualquier otro dato dentro de celdas individuales.

EJEMPLO: Creación de un arreglo celda

```
>> C = { 1, 2, 3; 'FIECS', randn(100,3), {11; 22; 'UNI'} }
C =
    [    1]    [           2]    [       3]
    'FIECS'    [100x3 double]    {3x1 cell}

>> whos C
  Name      Size             Bytes  Class    Attributes
  C         2x3               2996  cell
```

EJEMPLO: Acceso a datos en un arreglo celda

```
>> A = C(2,:)
A =
    'FIECS'    [100x3 double]    {3x1 cell}

>> whos A
  Name      Size             Bytes  Class    Attributes
  A         1x3               2792  cell

>> M = C(2,1)
M =
    'FIECS'

>> N = C{2,1}
N =
  FIECS

>> whos M N
  Name      Size             Bytes  Class    Attributes
  M         1x5                 10  char
  N         1x5                 10  char

>> Q = C(2,3)
Q =
       {3x1 cell}

>> R = C{2,3}
R =
       [11]       [22]       'UNI'

>> T = R(3)
T =
       'UNI'

>> S = R{3}
S =
UNI

>> V = C{2,3}(3)
V =
    'UNI'

>> W = C{2,3}{3}
W =   UNI

>> whos Q R T S V W
  Name      Size             Bytes  Class    Attributes
```

```
  Q         1x1               262  cell
  R         3x1               202  cell
  S         1x3                 6  char
  T         1x1                66  cell
  V         1x1                66  cell
  W         1x3                 6  char
```

EJEMPLO: Paso del contenido de arreglos celda a funciones

```
>> randCell = {'Random Data', rand(20,2)};
>> plot(randCell{1,2})
>> title(randCell{1,1})
>> figure
>> plot(randCell{1,2}(:,1))
>> title('First Column of Data')
```

5.2. Funciones Avanzadas

5.2.1. Manipuladores de Función (function handle)

Un manipulador de función es un valor MATLAB que provee un mecanismo de invocación indirecta de una función. Se puede pasar manipuladores de función en invocaciones a otras funciones (tambien llamadas funciones de función). También se puede almacenar los manipuladores de función en estructuras de datos para un posterior uso (por ejemplo, los manipuladores de callbacks gráficos). Un manipulador a función es uno de los tipos de datos estándar de MATLAB.

Al mismo tiempo que sea crea un manipulador de función, la función que se especifica debe estar en la ruta de MATLAB y en el actual alcance (ámbito) del código en el que se crea el manipulador. Por ejemplo, se puede crear un manipulador a una función local siempre que se haga dentro del archivo que define la función local. Esta condición no se aplica cuando se evalúa el manipulador de función. Se puede, por ejemplo, ejecutar una función local desde un archivo separado (fuera del alcance) usando un manipulador de función. Esto requiere que el manipulador sea creado mediante la función local (dentro del alcance).

`handle = @(lista_de_argumentos) funcion_anonima` construye una función anónima y retorna un manipulador a dicha función.

- El cuerpo de la función, a la derecha del paréntesis, es una sentencia MATLAB o comando.
- `lista_de_argumentos` es una lista separada por comas de argumentos de entrada.
- La función se ejecuta invocandola mediante el manipulador de funcion, `handle`.

EJEMPLO: Construcción de un manipulador a una función nombrada (definida en un archivo M o predefinida en MATLAB)

```
1  function x = procesoar1(phi, y0, sigmae, T)
2
3  % vector de innovaciones
4  e = sigmae*randn(T,1);
5
6  % preasignacion de espacio en memoria
7  x = zeros(T,1);
8
9  % creacion del proceso AR1
10 x(1) = phi*y0 + e(1);
11 for t=2:T
12   x(t) = phi*x(t-1) + e(t);
13 end
```

► **Ejecución:**

```
>> y1 = procesoar1(0.9, 0, 1, 100);
>> y2 = feval('procesoar1', 0.98, 0, 1, 200);
>> h = @procesoar1
h =
    @procesoar1
>> y3 = feval(h, 0.96, 0, 1, 200);
>> y4 = feval(@procesoar1, 0.99, 0, 1, 300);
>> whos h y1 y2 y3 y4
  Name        Size            Bytes  Class              Attributes
  h           1x1                16  function_handle
  y1        100x1               800  double
  y2        200x1              1600  double
  y3        200x1              1600  double
  y4        300x1              2400  double
```

EJEMPLO: Construcción de un manipulador a una función anónima

```
>> pow = @(x,n) x.^n
pow =
    @(x,n)x.^n

>> y = pow( [2 3 4], 2)
y =
     4     9    16
```

EJEMPLO: Construcción de una función que recibe como argumento de entrada una función mediante el nombre del archivo M en el que esta modelada o a través de un manipulador a ella o a través de una función anónima.

■ `trapecio.m` (función)

```
1 function I = trapecio(strfuncion, x1, x2, n)
2 f1 = feval(strfuncion, x1);
3 f2 = feval(strfuncion, x2);
4 h = (x2-x1)/n;
5 S = sum( feval(strfuncion, x1+(1:n-1)'*h)  );
6 I = (h/2)*(f1 + 2*S + f2);
```

■ `mifuncion.m` (función)

```
1 function y = mifuncion(x)
2 y = x.*exp(-x.^2);
```

► **Ejecución:**

```
>> I = trapecio('mifuncion', 0, 5, 100)
I =
    0.4998

>> h = @trapecio
h =
   @trapecio

>> h = @mifuncion
h =
   @mifuncion

>> I = trapecio(h, 0, 5, 100)
```

```
I =
   0.4998

> I = trapecio(@mifuncion, 0, 5, 100)
I =
   0.4998

>> fcn = @(x) x.*exp(-x.^2)
fcn =
   @(x)x.*exp(-x.^2)

>> I = trapecio(fcn, 0, 5, 100)
I =
   0.4998
```

Observaciones:

- Un manipulador de función es un tipo de dato MATLAB estándar. Como tal, se puede manipular y operar con manipuladores de funciones de la misma manera que con otros tipos de dato MATLAB. Esto incluye el uso de manipuladores de funciones en arreglos estructura y celda:

  `S.a = @sin; S.b = @cos; S.c = @tan;`

  `C = {@sin, @cos, @tan};`

  Sin embargo, las matrices estándar o arreglos de manipuladores de funciones no son soportados:

  `A = [@sin, @cos, @tan];% Esto no es soportado`

- Use `isa(h, 'manipulador_de_funcion')` para verificar si `h` es un manipulador de función

5.2.2. Funciones Locales (subfunciones)

Los programas MATLB puede contener código con mas de una función.

La primera función en el archivo es conocida como la *función principal*, es visible por las funciones desarrolladas en otros archivos y pueden ser invocadas desde la línea de comando. Las funciones adicionales dentro del archivo son llamadas *funciones locales*. Las *funciones locales* son solo visibles por otras funciones en el mismo archivo. Ellas son el equivalente a las *subrutinas* en otros lenguales de programación, y son comúnmente llamadas como *subfunciones*.

Las funciones locales pueden ocurrir en cualquier orden, siempre y cuando la función principal aparezca primero.

EJEMPLO:

■ `misestadisticos.m` (función)

```
1  function [promedio, mediana] = misestadisticos(x)
2    n = length(x);
3    promedio = mimedia(x,n);
4    mediana = mimediana(x,n);
5  end
6
7  function a = mimedia(v,n)
8    a = sum(v)/n;
9  end
10
11 function m = mimediana(v,n)
12   w = sort(v);
13   if rem(n,2) == 1
14     m = w((n + 1)/2);
15   else
16     m = (w(n/2) + w(n/2 + 1))/2;
```

```
17    end
18  end
```

5.2.3. Funciones Anidadas

Una *función anidada* (*nested*) es una función que esta completamente contenida dentro de una función padre. Cualquier función en un archivo de programa puede incluir una función anidada.

Las funciones anidadas se diferencias de los otros tipos de funciones en que ellas pueden acceder y modificar variables que estan definidas en sus funciones padres. Como resultado de esto, se tiene que:

- Las funciones anidadas pueden usar variables que no son explícitamente pasadas como argumentos de entrada.
- En la función padre, se puede crear un manipulador a una función anidada que contenga los datos necesarios para ejecutar la función anidada.

Requerimientos

- Típicamente, las funciones no requieren una sentencia end. Sin embargo, para anidar cualquier función en un archivo de programa, todas las funciones en el archivo deben usar la sentencia end al finalizar.
- No se puede definir un función anidada dentro de alguna sentencia de control.
- Se debe invocar a una función anidada ya sea directamente por su nombre (sin el usar feval), o usando un manipulador de función creado usando el operador @ (y sin `str2func`).
- Todas las variables en las funciones anidadas o las fuciones que las contengan deben estar explícitamente definidas. Esto es, no se pude invocar a una función o script que asigne valores a variables a menos que dichas variables ya existan en el workspace de la función.

Compartición de variables entre la función padre y la anidada

En general, las variables en el workspace de una función no estan disponibles a otras funciones. Sin embargo, las funciones anidadas pueden acceder y modificar variables en los workspaces de las funciones que las contengan.

Esto significa que ambas, una función anidada como una función que la contenga, pueden modificar la misma variable sin pasarse dicha variable como si un argumento.

EJEMPLO: En cada una de las siguientes funciones, `main1` y `main2`, ambas funciones principales y la función anidada puede acceder a la variable `x`:

```
function main1
  x = 5;
  nestfun1;

  function nestfun1
    x = x + 1;
  end

end
```

```
function main2
  nestfun2;

  function nestfun2
    x = 5;
  end

  x = x + 1;
end
```

Cuando las funciones padre no usan una variable dada, la variable se mantiene local a la función anidada

EJEMPLO: En esta función llamada main, las dos funciones anidadas tienen sus propias versiones de x que no pueden interactuar una con la otra:

```
function main
  nestedfun1;
  nestedfun2;

  function nestedfun1
    x = 1;
  end

  function nestedfun2
    x = 2;
  end
end
```

Las funciones que retornan argumentos de salida tienen variables para las salidas en sus workspaces. Sin embargo, las funciones padre solo tienen variables para la salidas de las funciones anidadas si ellas expresamente lo solicitan.

EJEMPLO: Esta función `parentfun` no tiene a la variable y en su workspace:

```
function parentfun
  x = 5;
  nestfun;

  function y = nestfun
    y = x + 1;
  end

end
```

Si se modifica el código tal como se indica a continuación, la variable `z` estará en el workspace de `parentfun`

```
function parentfun
  x = 5;
  z = nestfun;

  function y = nestfun
    y = x + 1;
  end

end
```

Uso de manipuladores para almacenar parámetro de funciones

Las funciones anidadas pueden usar variables de tres fuentes:

- Argumentos de entrada
- Variables definidas dentro de la función anidada
- Variables definidas en una función padre, también llamadas *variables externamente alcanzadas*.

Cuando se crea una manipulador de función para una función anidada, dicho manipulador almacena no solo el nombre de la función, sino tambien los valores de las variables externamente alcanzables.

EJEMPLO: Crear una función en un archivo llamado `makeParabola.m`. Esta función acepta varios coeficientes polinomiales, y retorna un manipulador a una función anidada que calcula el valor del polinomio.

```
function p = makeParabola(a,b,c)
  p = @parabola;
  z = nestfun;

  function y = parabola(x)
    y = a*x.^2 + b*x + c;
  end

end
```

La función `makeParabola` retorna un manipulador a la función `parabola` que incluye valores para los coeficientes `a`, `b` y `c`.

En la línea de comandos, llamamos a la función `makeParabola` con valores para los coeficientes de 1, 3, 2 y 30. Luego, usamos el manipulador de la función `p` para evaluar el polinomio en un punto en particular:

```
>> p = makeParabola(1.3,.2,30);
>> x0 = 25;
>> Y = p(x0)
Y =
  847.5000
```

Funciones MATLAB que usan manipuladores de función

Diversas funciones MATLAB aceptan entradas manipulador de función para evaluar funciones sobre un rango de valores.

EJEMPLO: Graficar la ecuación parabólica en el intervalo $x \in [-25; 25]$.

```
>> fplot(p,[-25,25])
```

Incluso, se pueden crear multiples manipuladores a la función parábola cada una con diferentes coeficientes polinomiales:

```
>> firstp = makeParabola(0.8,1.6,32);
>> secondp = makeParabola(3,4,50);
>> range = [-25,25];
>> figure
>> hold on
>> fplot(firstp,range)
>> fplot(secondp,range,r:)
>> hold off
```

Visibilidad de Funciones Anidadas.

Cada función tiene un cierto alcance, esto es, un conjunto de otras funciones desde las cuales es visible. Una función anidada es accesible:

- Desde el nivel inmediato superior.
- Desde una función anidada en el mismo nivel dentro de la misma función padre.
- Desde una función en cualquier nivel inferior.

Cuando se crea un manipulador de función para una función anidada, aquel manipulador almacena no solo el nombre de la función, sino también los valores de las variables externamente alcanzadas.

```
function A(x, y) % Funcion principal
  B(x,y);
  D(y);

  function B(x,y) % Anidado en A
    C(x);
    D(y);

    function C(x) % Anidado en B
      D(x);
    end
  end

  function D(x) % Anidado en A
    E(x);
    function E(x) % Anidado en D
      disp(x)
    end
  end
end
```

La forma mas sencilla de extender el alcance de una función anidada es creando un manipulador de función y retornandolo como un argumento de salida, tal como se muestra en el **uso de manipuladores para almacenar parámetro de funciones**.

5.2.4. Funciones con numero variable de argumentos

Para que permitir a las funciones aceptar un numero variable de argumentos de entrada y salida se utiliza `varargin` y `varargout` respectivamente

varargin

- Es una variable de entrada en sentencia de definición de una función que permite a la función aceptar cualquier numero de argumentos de entrada.
- Se debe especificar `varargin` con caracteres minúscula, e incluirla simpre al final de cualquier otra declaración explicita de argumentos de entrada.
- Cuando la función se ejectua, `varargin` es un *arreglo celda* de $1 \times N$, donde N es el número de entradas que la función recibe después de las entradas explícitamente declaradas.
- `nargin` retorna el numero de argumentos de entrada (N) pasados en la llamada a la actual función en ejecución. Se debe usar solo en el cuerpo de la función.

EJEMPLO: Definir una función en un archivo llamado `varlist.m` que acepte un numero variable de argumentos de entrada y muestre los valores de cada entrada.

■ `varlist.m` (función)

```
function varlist(varargin)
fprintf(Numero de argumentos: %d\n,nargin);
celldisp(varargin)
```

Luego, podemos invocar a varlist con varias entradas

```
>> varlist(ones(3), 'algun texto', pi)
Numero de argumentos: 3

varargin{1} =
 1 1 1
 1 1 1
 1 1 1

varargin{2} =
algun texto

varargin{3} =
3.141
```

varargout

- Es una variable de salida en sentencia de definición de una función que permite a la función aceptar cualquier numero de argumentos de salida.
- Se debe especificar `varargout` con caracteres minúscula, e incluirla simpre al final de cualquier otra declaración explicita de argumentos de salida.
- Cuando la función se ejectua, `varargout` es un *arreglo celda* de $1 \times M$, donde M es el número de salidas que la función recibe después de las salidas explícitamente declaradas.
- `nargout` retorna el numero de argumentos de salida (M) pasados en la llamada a la actual función en ejecución. Se debe usar solo en el cuerpo de la función.

EJEMPLO: Definir una función en un archivo llamado `sizeout.m` que retorne el tamaño del vector de salida s y un numero variable de valores escalares adicionales.

■ `sizeout.m` (función)

```
1 function [s,varargout] = sizeout(x)
2 nout = max(nargout,1) - 1;
3 s = size(x);
4 for k=1:nout
5   varargout{k} = s(k);
6 end
```

El argumento de salida `s` contiene las dimensiones del arreglo de entrada `x`. Los argumentos de salida adicionales corresponden corresponden a las dimensiones individuales dentro de `s`.

Invocando a `sizeout` con un arreglo tridimensional y exigiendo tres salidas.

```
>> [s,rows,cols] = sizeout(rand(4,5,2))
s =
     4 5 2

rows =
     4

cols =
     5
```

Capítulo 6

Modelamiento de Sistemas Dinámicos con Simulink

6.1. Simulink

Simulink es una herramienta que ofrece un editor gráfico, bibliotecas de bloques personalizables y un conjunto de solvers, para modelar y simular **sistemas dinámicos**. Esta basado en un entorno de diagramas de bloque multidominio bajo un diseño basado en modelos. Simulink permite el diseño y la simulación a nivel de sistema, la generación automática de código, así como la prueba y verificación continua de los sistemas embebidos.

La capacidad de integración de Simulink con MATLAB, le permite incorporar algoritmos de este lenguaje dentro de los modelos Simulink, exportar los resultados de la simulación a MATLAB para así poder llevar a cabo mas análisis.

Dentro del entorno MATLAB, Simulink es un toolbox que se diferencia de los otros, tanto por su interfaz especial como por la "técnica de programación". El código fuente del Sistema Simulink no es abierto.

Los **sistemas dinámicos** pueden ser simulados utilizando Simulink. En la mayoría de casos, estos implican procesos lineales o no lineales dependientes del tiempo, que pueden ser descritos usando ecuaciones diferenciales (tiempo continuo) o ecuaciones en diferencia (tiempo discreto). Otra forma común de describir los sistemas dinámicos es mediante los **diagramas de bloque**.

Los **diagramas de bloques** es un intento de entender el comportamiento del sistema por medio de una representación gráfica, que esencialmente consiste de representaciones de los componentes individuales del sistema (bloques) junto con un flujo de señales entre estos componentes. Simulink se basa en esta forma de representación, para ello usa una interfaz gráfica para convertir un diagrama de bloques de esta clase (casi) directamente en un modelo Simulink y luego simular el funcionamiento del sistema. Hay que observar que un uso bien fundamentado de Simulink requiere ciertos conocimientos de tecnología de **control** y la teoría de **sistemas**, por lo que a nivel introductorio nos limitaremos a un tema central, la *solución numérica de simples ecuaciones diferenciales.*

Como se señaló anteriormente, los sistemas dinámicos (que son continuos en el tiempo) se describen por ecuaciones diferenciales. Por lo tanto, cuando se describe el sistema con un diagrama de bloques y se simula la reacción del sistema a una señal de entrada, esencialmente estamos buscando nada más que la *solución de la ecuación diferencial en la que se basa el sistema.*

También es posible convertir una ecuación diferencial en un diagrama de bloques y resolver numéricamente la ecuación diferencial con Simulink. Simulink es, por lo tanto, un solucionador numérico (solver) de ecuaciones diferenciales.

Las Características Principales de Simulink son:

- Editor gráfico para crear y gestionar diagramas de bloques jerárquicos.
- Bibliotecas de bloques predefinidos para modelar sistemas continuos y discretos.
- Motor de simulación con solvers de ecuaciones diferenciales ordinarias de paso fijo y paso variable.
- Scopes y data displays para ver los resultados de la simulación.
- Herramientas de gestión de proyectos y datos para administrar los archivos y los datos del modelo.

- Herramientas de análisis de modelos para perfeccionar la arquitectura del modelo y aumentar la velocidad de simulación.
- Bloque MATLAB Function para importar algoritmos de MATLAB en modelos.
- Legacy Code Tool para importar código C y C++ a los modelos.

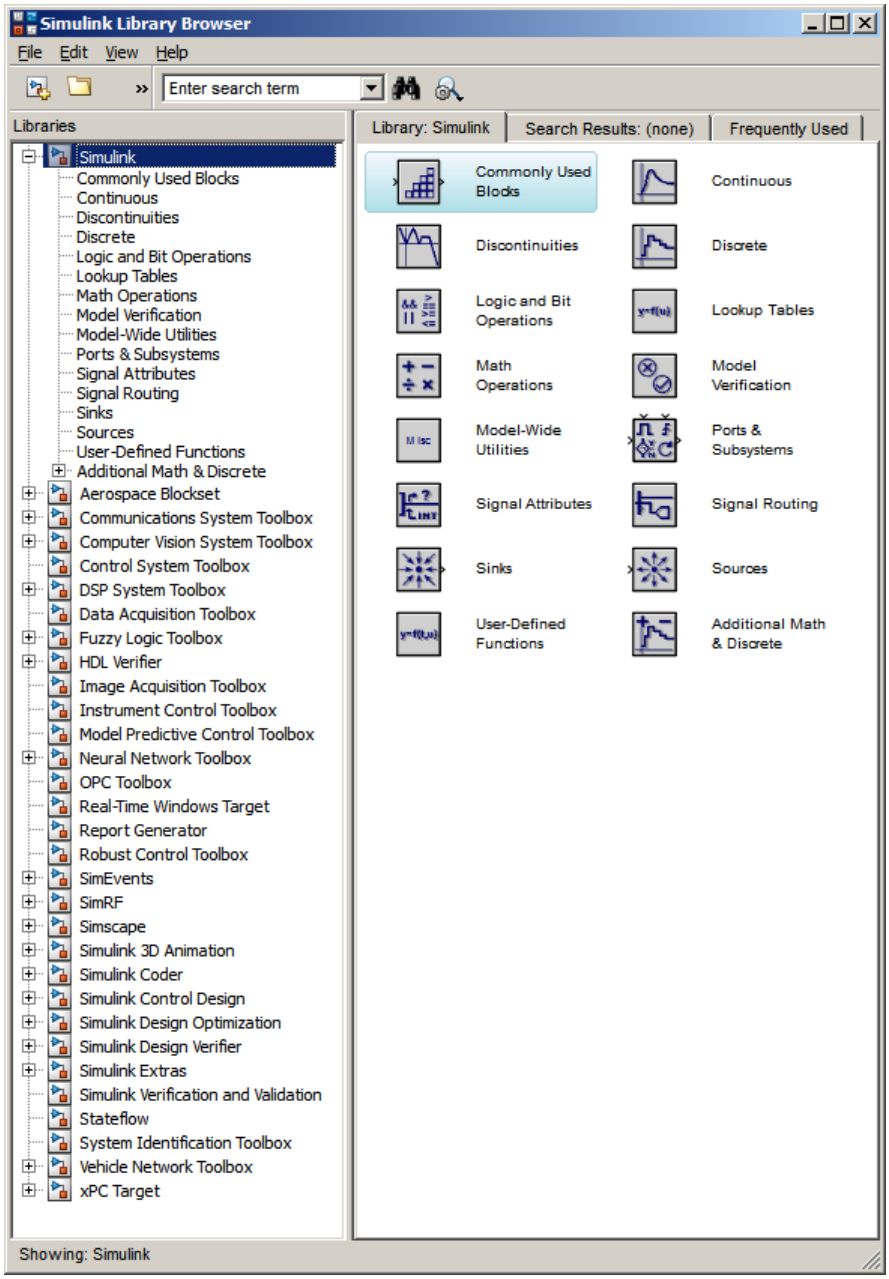
6.2. Principios de Operación y Gestión de Simulink

El programa se inicia desde la ventana de comandos de MATLAB. A continuación mostramos tres formas de iniciar Simulink.

1. Con el comando

   ```
   >> simulink
   ```

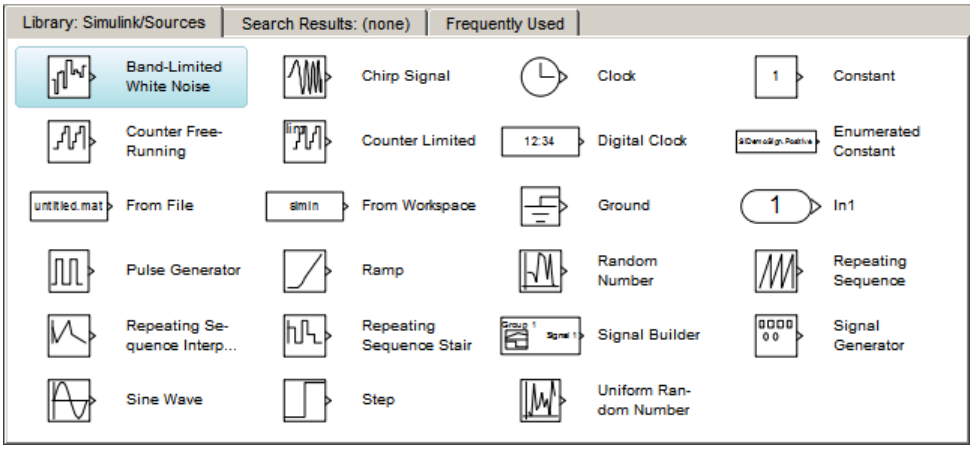
   aperturándose a continuación el Simulink Library Browser



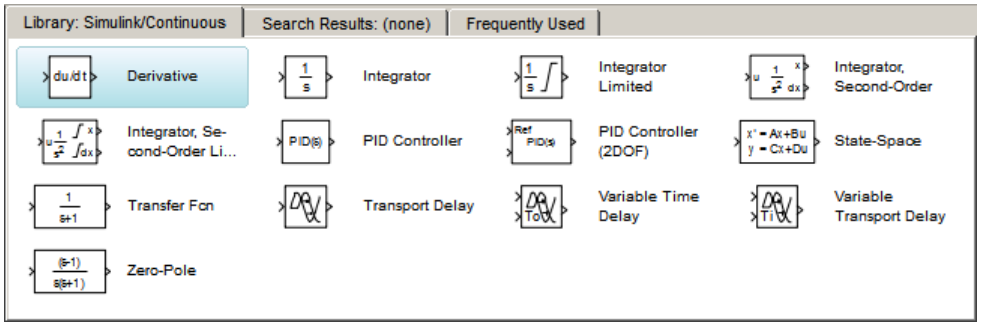
El **Simulink Library Brower (SLB)** visualiza las bibliotecas de bloques disponibles (según la instalación) organizadas en grupos funcionales, los cuales a su vez pueden contener subgrupos.

Por defecto, el SLB se posiciona en la biblioteca **Simulink**, la cual contiene:

- La biblioteca **Sources**, conteniendo bloques para la producción de señales (funciones):



- La biblioteca **Continuous**, conteniendo bloques básicos para el tratamiendo de Señales en tiempo contínuo.

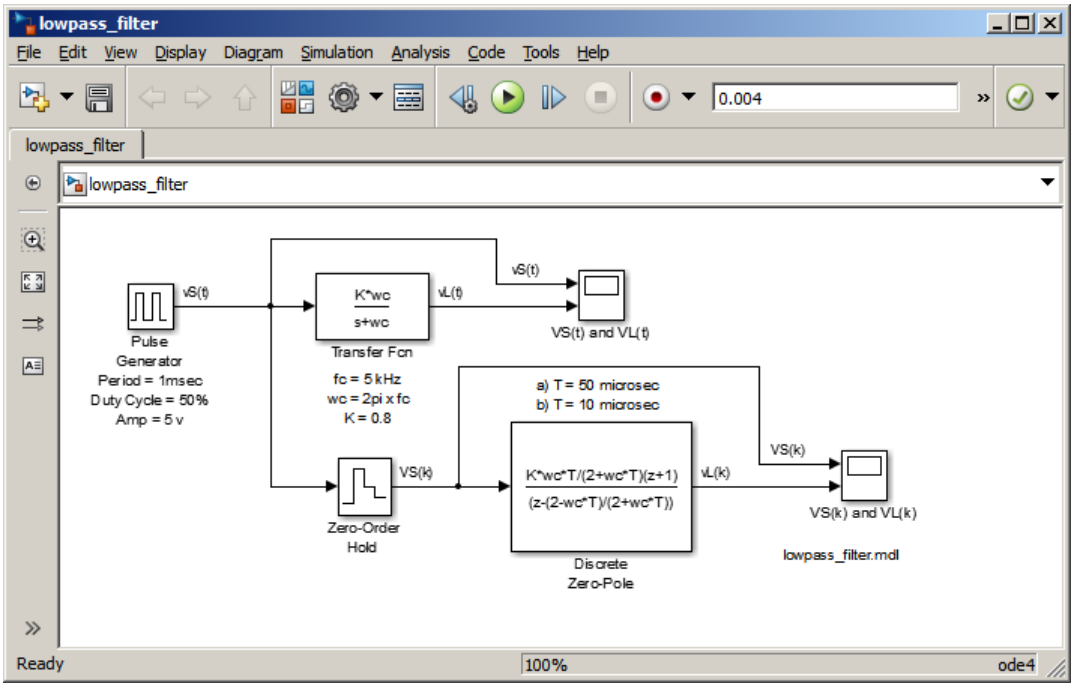


y muchas otras más.

2. Si contamos con un archivo modelo, por ejemplo `'lowpass_filter.mdl'`, usaremos el comando

   ```
   >> open_system('lowpass_filter.mdl')
   ```

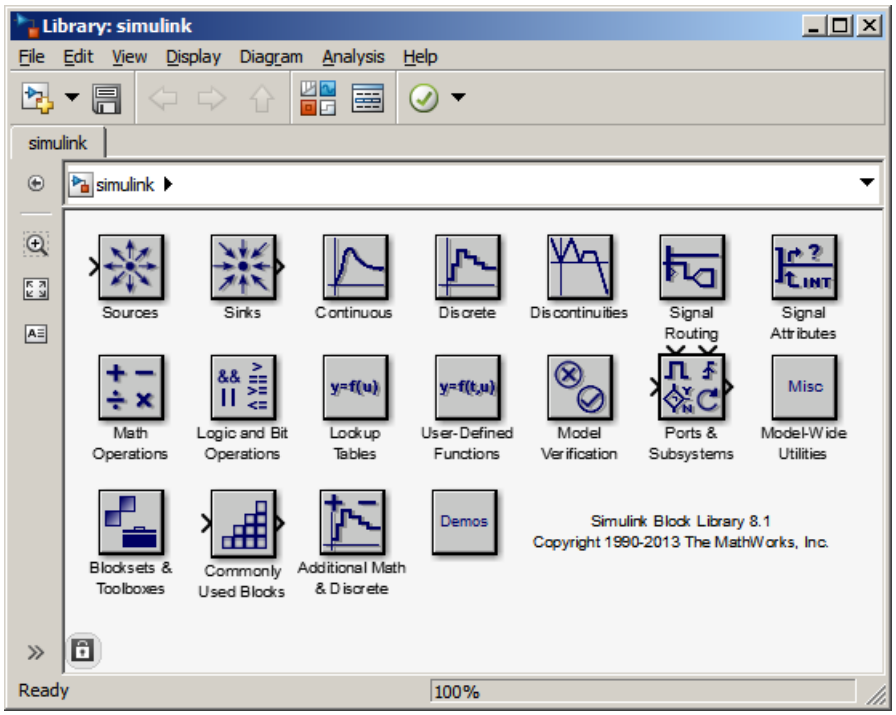
   cargándose en memoria y visualizándose gráficamente el modelo Simulink del sistema dinámico que representa.



3. Si se desea solo trabajar con la biblioteca **Simulink**, usaremos el comando

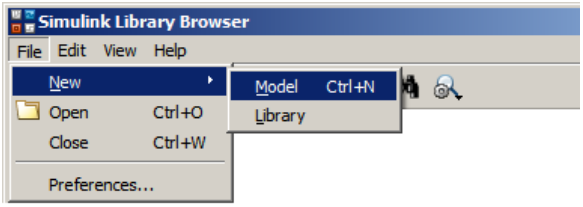
   ```
   >> open_system('simulink.mdl')
   ```

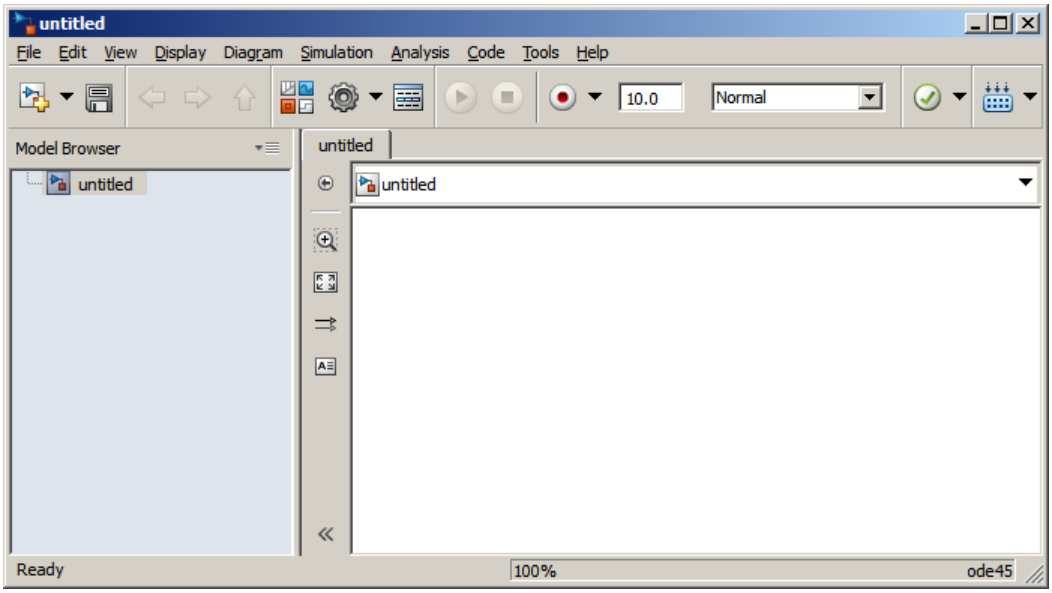
   apareciendo una ventana en la cual los símbolos para las diferentes clases de bloques de funciones son solo visulizadas en forma de íconos.



6.2.1. Construcción de un Diagrama de Bloques Simulink

Si se desea crear nuestro propio sistema de simulación usando las bibliotecas de bloque, primero se tiene abrir una ventana vacía seleccionando la opción `File|New` Model en el SLB.





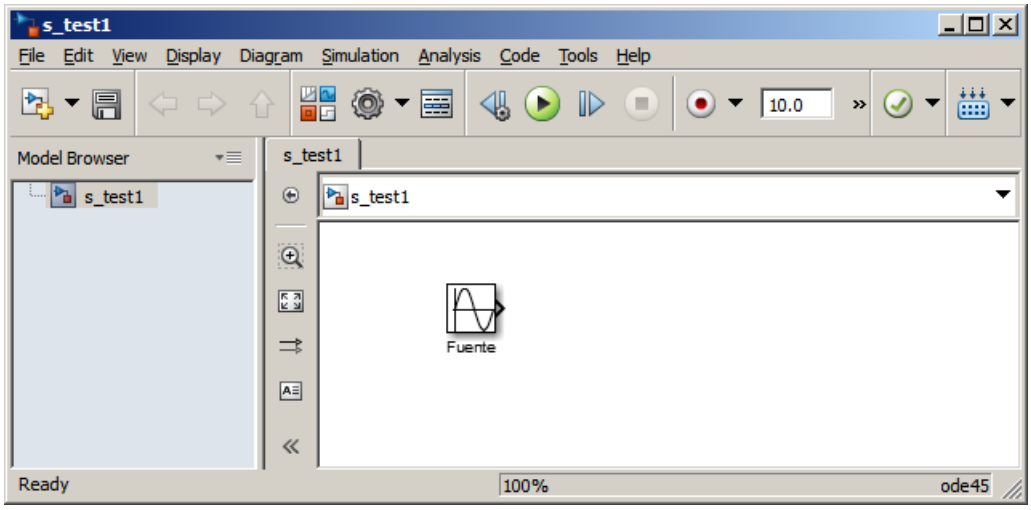
Los modelos (diagramas de bloques) ya existentes pueden ser abiertos bajo sus nombres de archivo seleccionando la opción `File | Open`. Es recomendable que una ventana vacia sea guardada inmediatamente con algún nombre de archivo adecuado como un archio mdl (mdl=model) usando la opción `File|Save As`.

EJEMPLO: Crear un modelo con los bloques que permitan graficar una señal senoidal y obtenga su integral.

Para esto realizaremos los siguientes pasos:

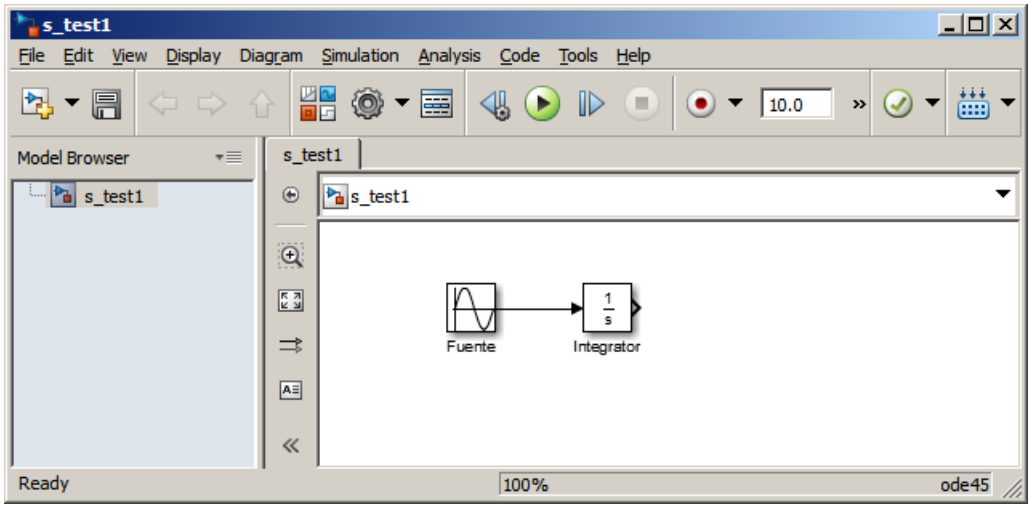
1. Iniciamos Simulink
2. Creamos un Nuevo Modelo
3. Guardamos el nuevo modelo con el nombre de archivo **s_test1.mdl**
4. Con el mouse arrastramos el bloque **Simulink|Sources|Sine Wave** fuera del SLB hacia a la ventana vacia. Si no se desea usar el nombre "sine wave", entonces se puede dar clic con el mouse en la línea de texto "sine wave" y editar el nombre con el teclado. De esta manera renombramos el bloque como "Fuente".

   El sistema **s_test1.mdl** tendrá una forma como la siguiente



5. Añadir el bloque Integrator que permita integrar la señal de salida del bloque Fuente

   Para esto, abrimos la biblioteca de funciones **Continous**. Arrastramos el bloque Integrator a partir de esta biblioteca a la ventana de **s_test1** y usando el mouse conectamos la salida del bloque "**Fuente**" a la entrada del bloque **Integrator** . Al inicio la habilidad para hacer la conexión toma algo de práctica.



NOTA: Es siempre mejor dibujar la línea de conexión en sentido opuesto a la dirección de propagación de la señal desde la entrada del bloque objetivo hacía la salida del bloque fuente; esto es, desde el Integrador hacia la Señal Fuente en este ejemplo.

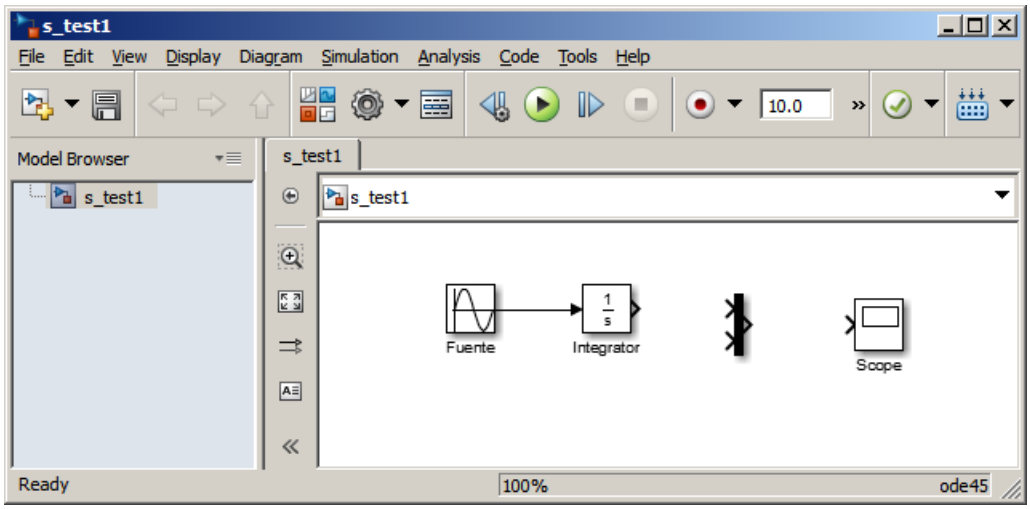
NOTA: Obseve que la entrada $\frac{1}{s}$ en el bloque Integrator, esta relacionada con la transformada de Laplace de la integración. Muchos de los bloques de funciones lineales están caracterizados por la transformada de Laplace o la Transformada Z (la contraparte discreta de la transformada de Laplace).

6. Extender el sistema de prueba **s_test1** de tal manera que la señal senoidal y su integral puedan ser vistas en una única ventana.

   Para hacer esto:

   a) Elegimos el bloque Simulink|Signal Routing|Mux y lo añadimos al sistema.

   b) Elegimos el bloque Simulink|Sinks|Scope y lo añadimos al sistema.



   c) Luego, realizamos las siguientes interconexiones:

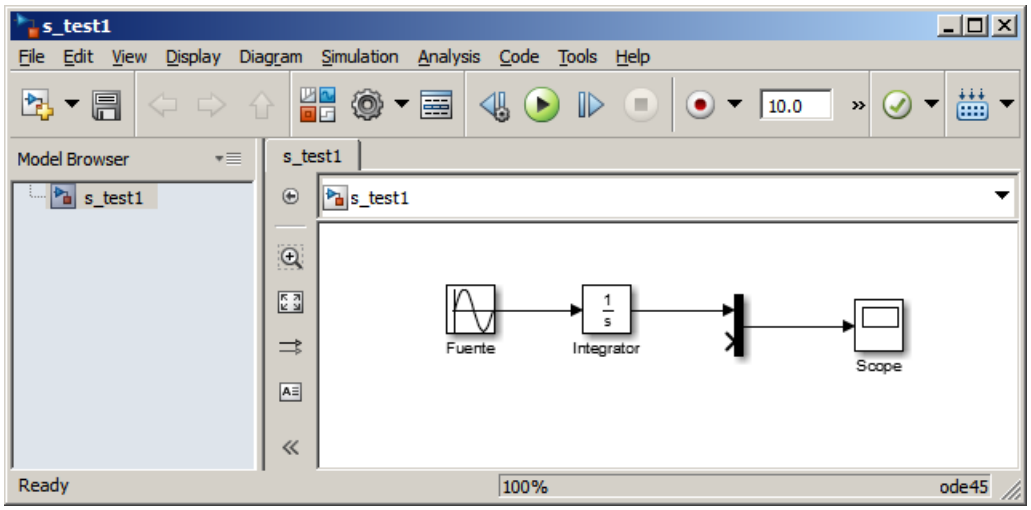
      1) La salida del bloque Señal Fuente con la entrada del bloque Integrator,
      2) La salida del bloque Integrator con la primera entrada del bloque Mux, y
      3) La salida del bloque Mux con la entrada del bloque Scope.

   NOTA: Conexión entre bloques (Interconexión: Bloque-Bloque)

   El puntero del mouse posee el aspecto cotidiando de una flecha (selección normal). Cuando se desea hacer una interconexión de bloques, el puntero del mouse debe ser dirigido hacia alguno de los puertos de entrada o salida de algun bloque Simulink y sólo cuando cambia su aspecto al de una cruz + (selección precisa), es que cualquier evento (clic izquierdo, clic derecho, doble clic, etc.) que se dé en ese instante estará asociado a dicho puerto de entrada o salida.

   Para realizar una conexión en Simulink lo que se debe hacer es dirigir el mouse hacía algún puerto (de entrada o salida) de algún bloque de interés, constantar que el puntero del mouse adopte la forma de una cruz, y en ése instante dar clic izquierdo del mouse y sin dejar de presionar el botón izquierdo dirigir el puntero hacia el puerto (de entrada o salida) o alguna señal (línea de interconexión), con quien deseamos establecer una nueva interconexión, durante este proceso la línea de interconexión (incompleta aún) se mostrará con guiones rojos (lo cual indica que la conexión aun no esta terminada o no es reconocida) y sólo cuando se muestre como una línea negra continua es que la conexión ya es reconocida y recién podemos soltar el botón izquierdo del mouse para definir la nueva interconexión.

   Tomando en cuenta la nota anterior, procedemos a realizar las tres interconexiones solicitadas

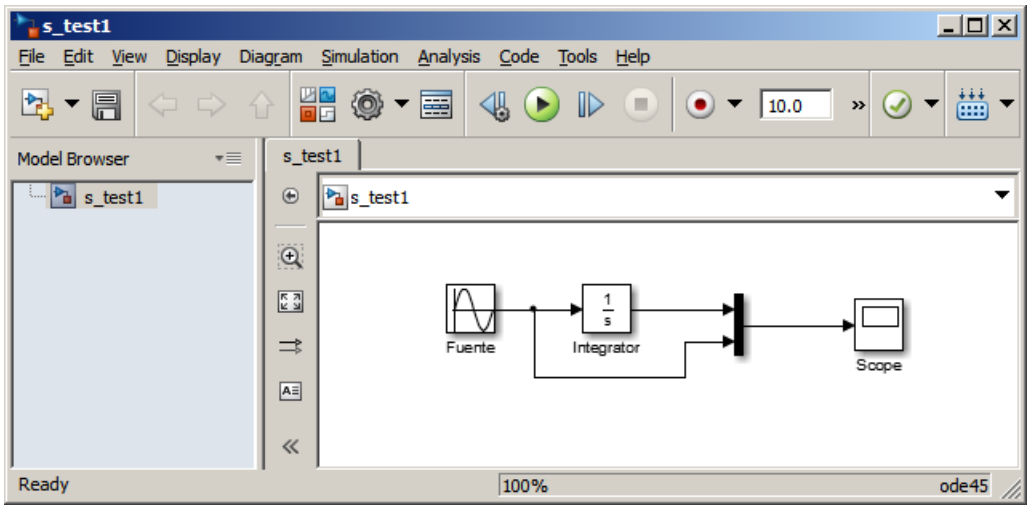


d) Finalmente, pasamos la señal de salida del bloque Fuente a la segunda entrada del bloque Mux.

NOTA: Paso de una señal a un bloque (Interconexión: Señal-Bloque)

Sólo cuando se desea que una señal de salida transmitida por medio de una línea de inerconexión hacia otro bloque destino, sea "pasada" (sin que se deje de seguir enviando la misma señal hacia el bloque al que originalmente llega la señal) a un nuevo bloque destino, es que tenemos que desarrollar la conexión a partir del puerto de entrada del nuevo bloque destino hacia algún punto de la línea de interconexión de donde queremos obtener la señal.

Tomando en cuenta la nota anterior, procedemos a realizar las la interconexión del segundo puerto de entrada del bloque Mux con algún punto sobre la línea (portadora de la señal) que conecta los bloques Señal Fuente e Integrator



6.2.2. Parametrización de los Bloques Simulink y de la Simulación

Parametrización de los Bloques

Para asignar valores a los parámetros de cada bloque, deberemos de abrir la lista de parámetros del bloque dándo doble clic en el correspondiente símbolo del bloque.

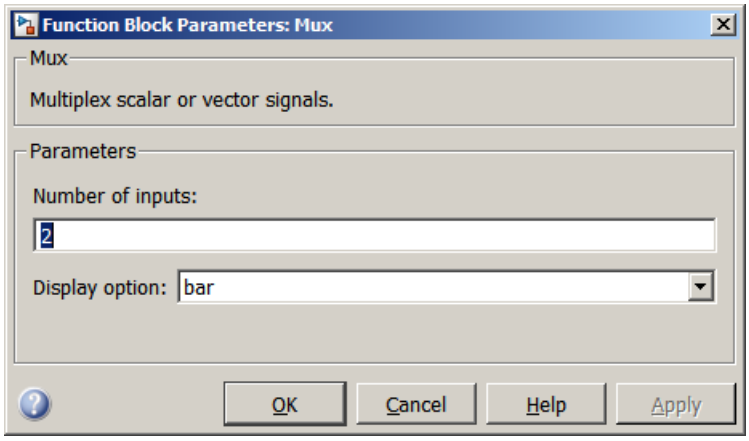
EJEMPLO: Crear un modelo con los bloques que permitan graficar una señal senoidal y obtenga su integral.

Para esto realizaremos los siguientes pasos:

1. Abra la lista de parámetros del bloque Mux y constante que:

   a) El parámetro **Number of inputs** esta predefinido en el valor 2.

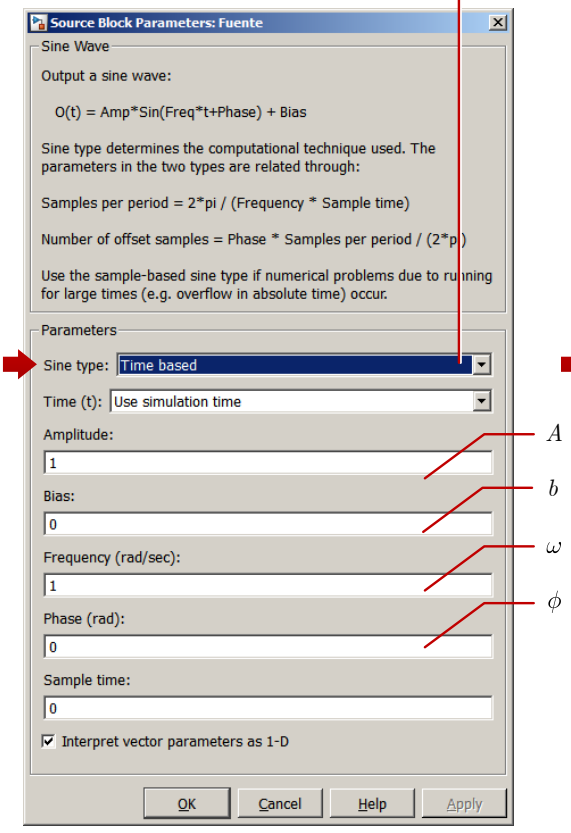
b) El parámetro **Display Options** esta predefinido en la opción **bar**.



2. Abra la lista de parámetros del bloque Fuente (Sine Wave) e identifique sus valores por defecto

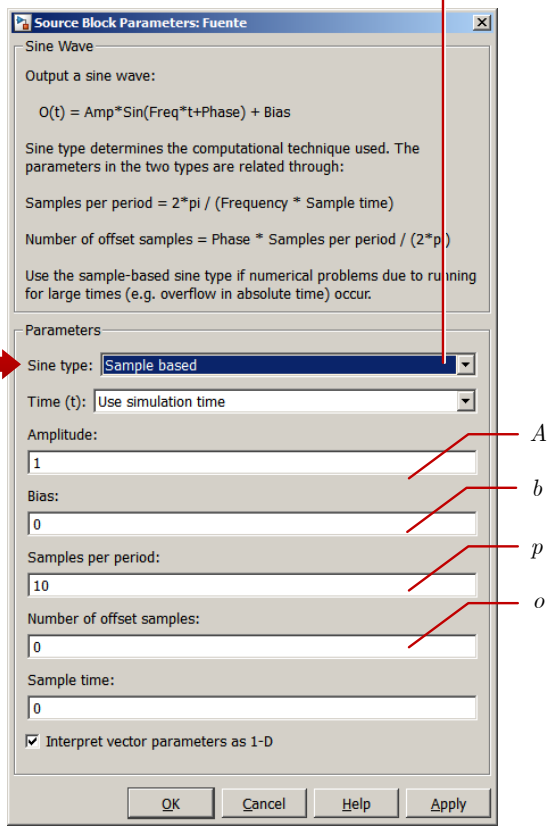
TIPO: Basado en el tiempo

$y(t) = A\sin(\omega t + \phi) + b$



TIPO: Basado en la muestra

$y(t) = A\sin(2\pi(k + o) / p) + b$



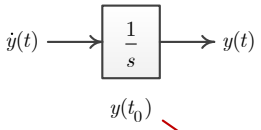
Luego, modifique los valores de los siguientes parámetros

- **Sine type:** Time based
- **Amplitude:** 2
- **Frequency (rad/sec):** 2*pi
- **Phase (rad):** pi/4

Los valores de los restantes parámetros quedan inalterados.

3. Abra la lista de parámetros del bloque Integrator e identifique sus valores por defecto.

Símbolo del Bloque Integrator

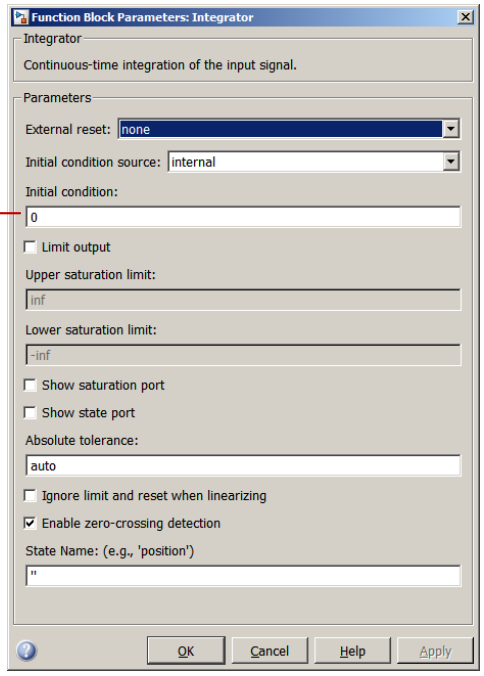


Ecuación del Bloque Integrator

$$\frac{dy}{dt} = \dot{y}(t)$$
$$dy = \dot{y}(t)dt$$
$$\int_{t_0}^{t} dy = \int_{t_0}^{t} \dot{y}(\tau)d\tau$$
$$y(t) - y(t_0) = \int_{t_0}^{t} \dot{y}(\tau)d\tau$$
$$\boxed{y(t) = y(t_0) + \int_{t_0}^{t} \dot{y}(\tau)d\tau}$$

donde t_0 : Start Time
$y(t_0)$: Initial Condition

Parámetros del Bloque Integrator



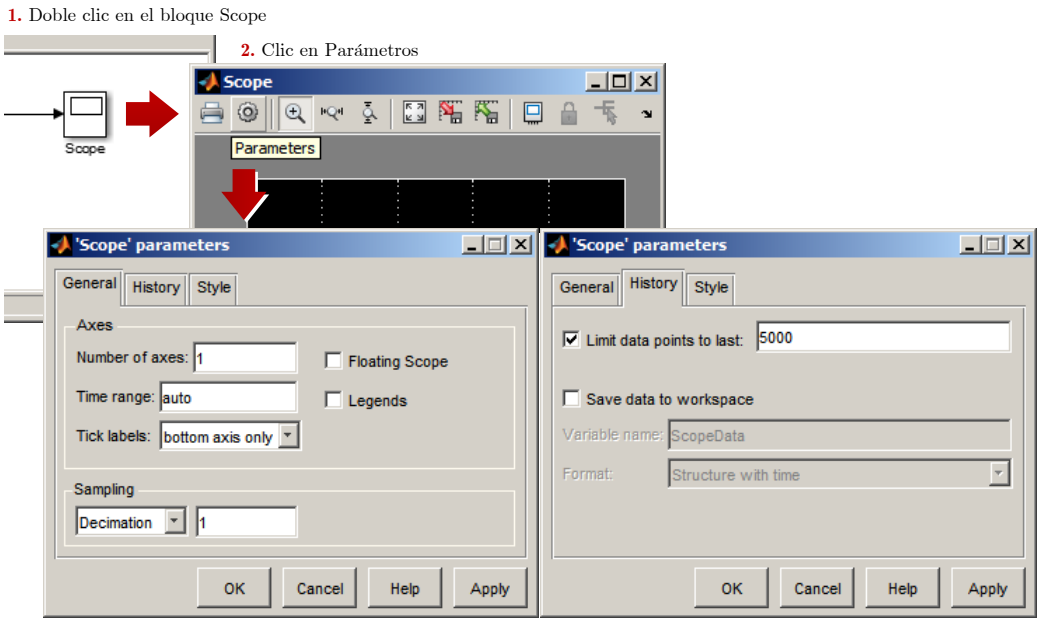
utilizando la notación de los manuales de Simulink, hacemos $u(t) = \dot{y}(t)$, quedando como ecuación del bloque

$$y(t) = \int_{t_0}^{t} u(\tau)\, d\tau + y(t_0)$$

En este caso, dejaremos todos los parámetros con sus valores por defecto. Nótese que por defecto la condición inicial es $y(t_0) = 0$.

4. Abra la lista de parámetros del bloque Scope e identifique sus valores por defecto.

Para acceder a los parámetros del bloque Scope deberemos en primer lugar, dar doble clic en el bloque Scope y luego dar clic en el botón Parameters. Los parámetros se visualizarán a continuación distribuidos en tres paneles General, History y Style. En la siguiente figura mostramos solo dos paneles.



En la pestaña **General**, se presentan dos paneles

- **Axes**, en donde se puede especificar en la propiedad **Number of axes** el número de entradas (ejes) que tendrá el bloque Scope (por defecto, 1), el rango de tiempo en la propiedad **Time Range**, y otras propiedades mas que serán detalladas en su momento.
- **Sampling**, será detallada más adelante.

En la pestaña History, se presentan dos casillas de verificación:

- **Limit data points to last** (por defecto activada) restringe la cantidad de puntos (observaciones de la muestra total) que se han de graficar en el bloque Scope a un valor determinado (5000 por defecto). En caso nuestra simulación exceda del límite solo se graficará la última parte tal que no exceda del maximo. En caso no deseemos esta restricción simplemente desactivamos la casilla.
- **Save data to workspace** (por defecto desactivada) nos brinda la posibilidad de que la señal de lectura del bloque Scope sea almacenada directamente como una variable de MATLAB. Para lo cual se deberá de activar e ingresar un nombre para la variable en la propiedad Variable name y especificar uno de los posibles formatos de escritura: Estructura con tiempo (por defecto), Estructura y Arreglo.

NOTA: La visualización de la señal en el bloque Scope es configurable solo después de la simulación a través de los botones de la barra de herramientas. Como el Scope no se abre automáticamente después de la simulación, éste debe ser abierto manualmente (después de la simulación).

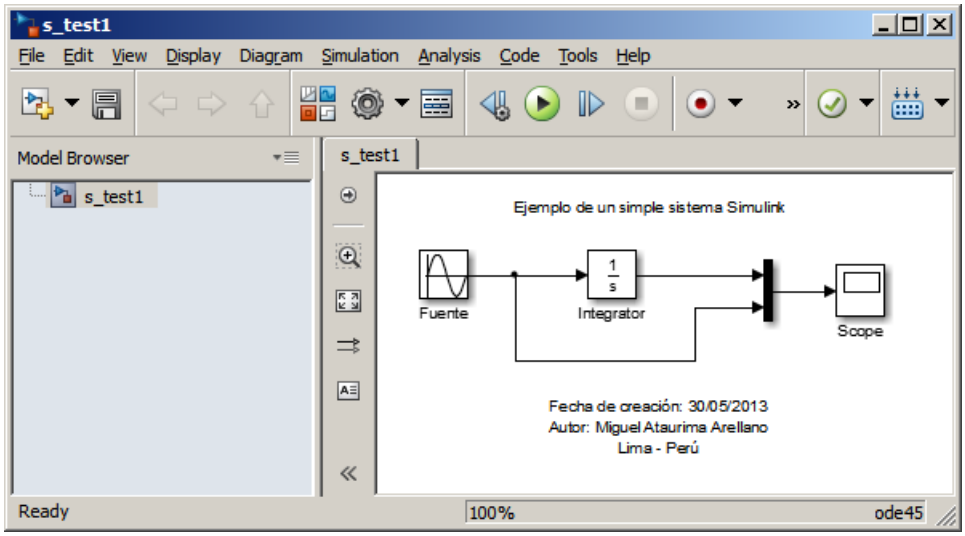
Ahora, modifique los valores de los parámetros de la pestaña History:

- **Limit data points to last:** Desactivado
- **Save data to workspace:** Activado
- **Variable Name:** S_test1_signals
- **Format:** Array

5. Añadir anotaciones al modelo

   Consiste en añadir texto de manera libre en el modelo de tal forma que podamos describirlo, esto es, colocar un título, sus detalles de su creación (autor, última fecha de modificación, etc.), etc. Es el equivalente a añadir comentarios (documentar) a un código fuente. Para lograr esto bastará con dar *doble clic* en la parte libre del modelo donde sea necesario insertar texto, y proceder a redactarlo.

   Por ejemplo, añadamos en el centro de la parte superior el título "Ejemplo de un simple sistema Simulink", y en el centro de la parte inferior datos de la fecha de creación y el autor.

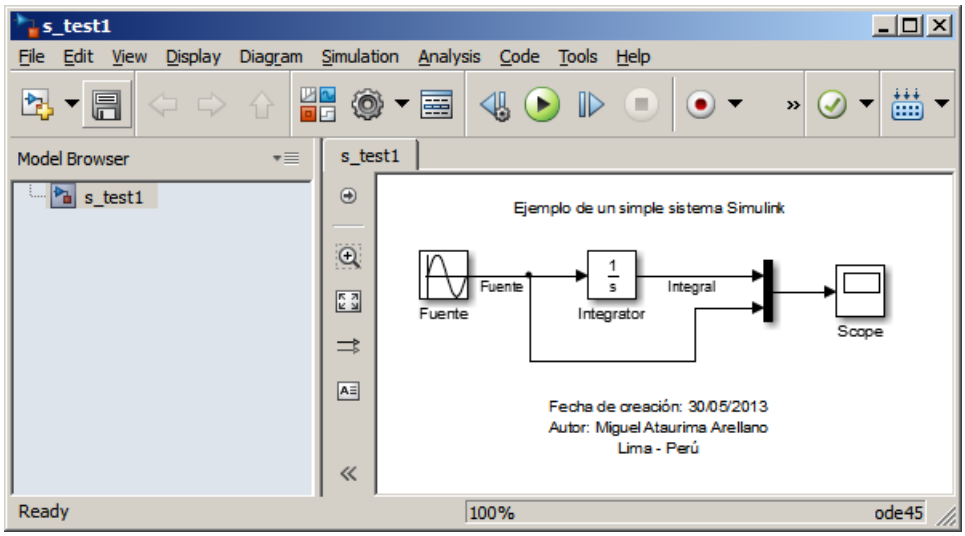


Ambos textos son editables, se puede arrastar con el mouse hacia alguna nueva posición deseada, pueden ser editados dándoles doble clic, y se le pueden especificar propiedades dándoles clic derecho y eligiendo la opción **Properties**.

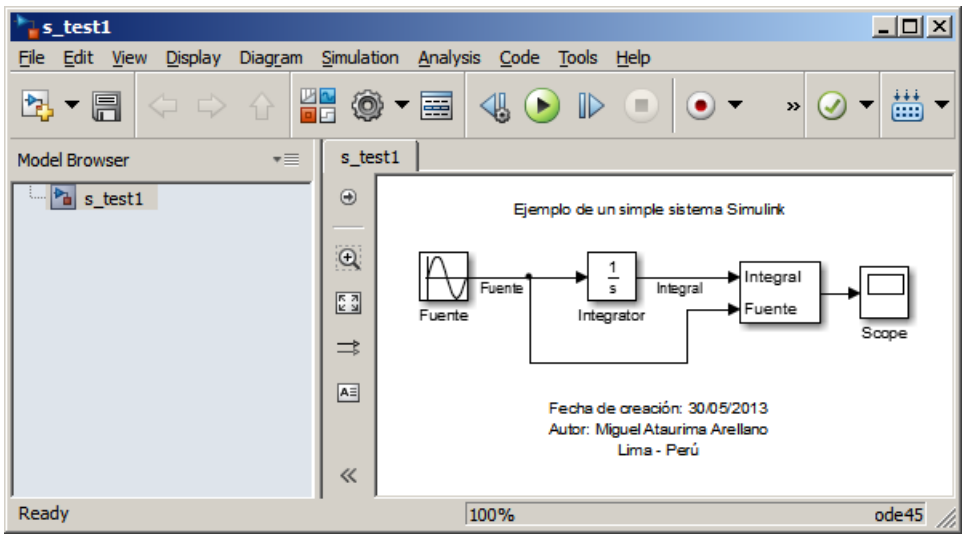
6. Etiquetar las señales

   Para ello bastará con dar doble clic sobre las flechas portadoras de señal y de inmediato digitar la etiqueta deseada. Estas etiquetas se desplazarán junto con las flechas portadoras de señal asociadas de manera automática.

   En nuestro ejemplo, asignemos la etiqueta "Fuente" a señal de salida del bloque Fuente y la etiqueta "Integral" a la señal de salida del bloque Integrator. En caso sea necesario desplace con el mouse el nodo que pasa la señal de la Fuente al bloque Mux.



   Luego, cambiamos el parámetro **Display Options** del bloque Mux al valor **signals**; luego, redimensionamos el bloque y observamos que las etiquetas de las señales son visualizadas en las entradas del bloque Mux.



7. Guardar los cambios realizados en el sistema (parametrización y anotaciones)

   Para ello eligiremos File|Save.

Parametrización de Simulación (Parámetros Generales)

Nos permitirá establecer valores para la duración de la simulación, los procedimientos de solución numéricos, etc. Esto lo lograremos eligiendo Simulation|Model Configuration Parameters.

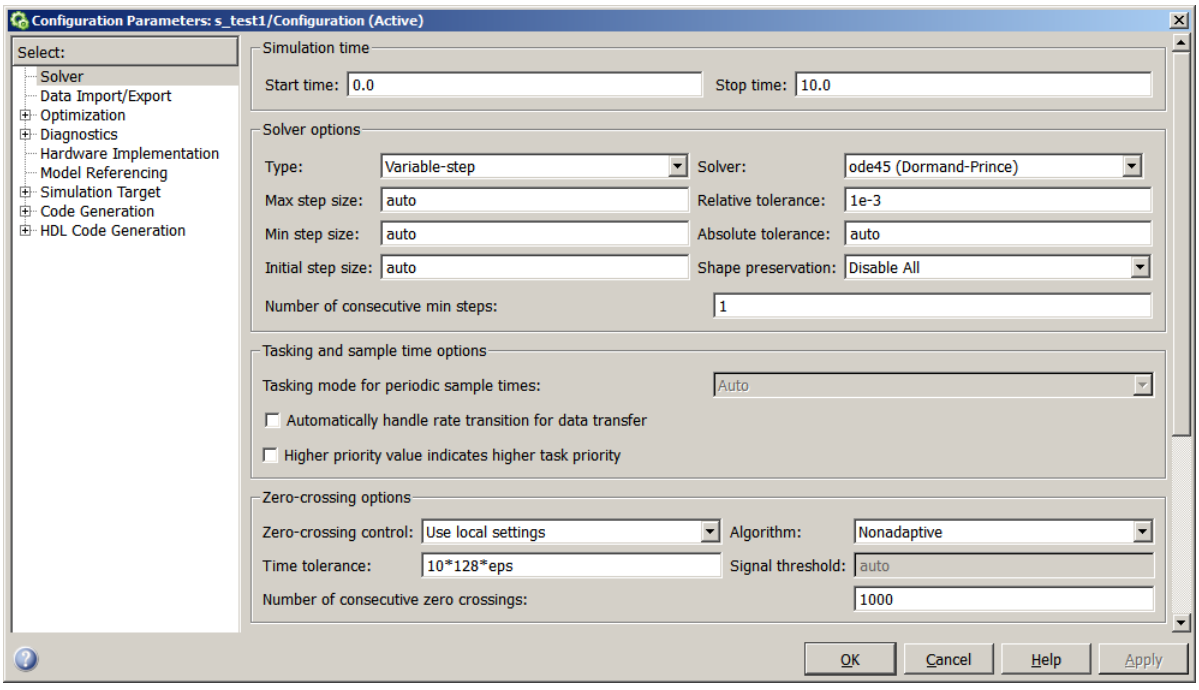
EJEMPLO: Establecer los siguientes parámetros de simulación al sistema creado en el ejemplo anterior.

- **Tiempo de Simulación** $[t_0, T]$ (en segundos)
  - Inicio de Simulación (t_0, Start Time) : 0
  - Finalización de Symulación (T, Stop Time): 20
- **Opciones del Solver**
  - **Tipo (Type)**
    Paso Fijo (Fixed-step)
  - **Solver (Método de íntegración numérica)**
    ode3 (Bogacki-Shampine[1])
  - **Tamaño del paso fijo**
    0.01

Para esto realizaremos los siguientes pasos:

1. Abrir la ventana de configuración de los parámetros del modelo

   Para ello elegimos Simulation|Model Configuration Parameters. Visualice los valores de los parámetros por defecto, grupo de parámetros **Solver**

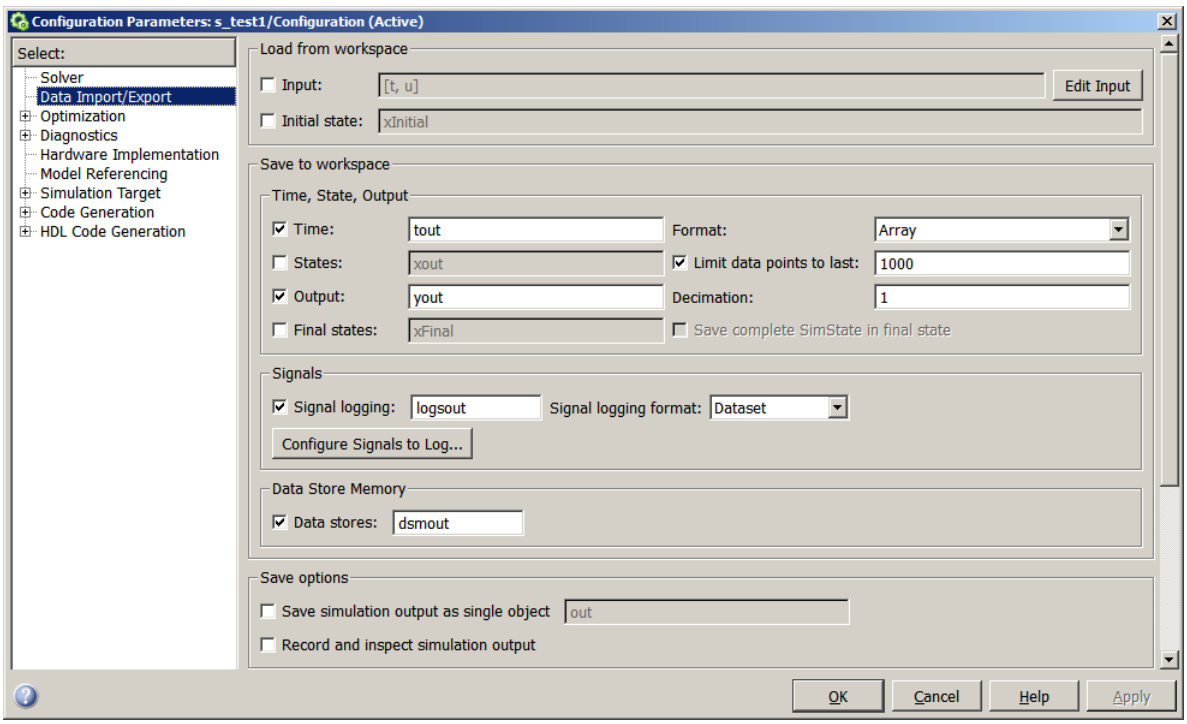


   Aquí es relevante distinguir lo siguiente:

   - En el *panel izquierdo* se presentan la vista de árbol **Select:**, que organiza la totalidad de parámetros requeridos para llevar a cabo la simulación del modelo. Por defecto, se tiene seleccionada la opción **Solver**.
   - En el *panel derecho* se visualizarán sub-paneles que agrupan, por categorías, a los parámetros pertenecientes a la opción seleccionada en el Panel Izquierdo. Por defecto, al tener seleccionada la opción **Solver**, se visualizarán los parámetros vinculados a dicha opción.

[1] El método Bogacki-Shampine es un método para la solución numérica de ecuaciones diferenciales ordinarias , que fue propuesta por Przemyslaw Bogacki y Lawrence F. Shampine en 1989 (Bogacki y Shampine 1989). El método Bogacki-Shampine es un **método de Runge-Kutta** de orden tres con cuatro etapas con la propiedad First Same As Last (FSAL), de modo que se utiliza aproximadamente tres evaluaciones de la función por paso. Tiene un método de segundo orden interno que puede ser utilizado para implementar un tamaño de paso adaptativo.

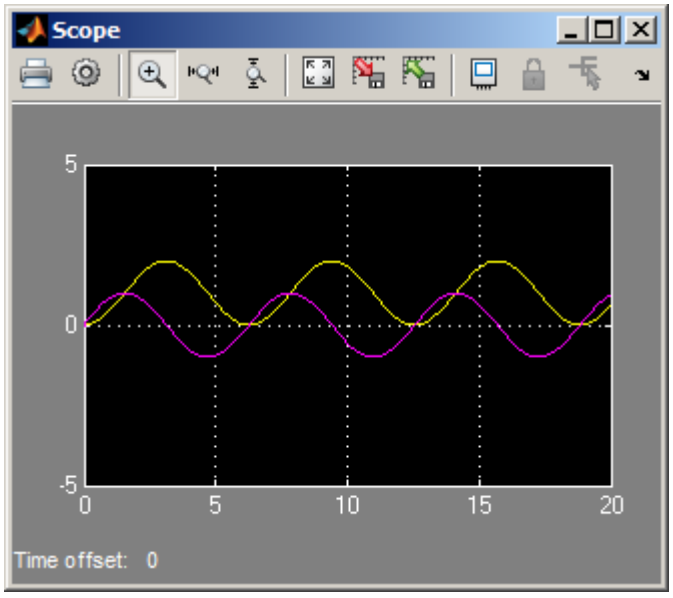
Luego, elegir el grupo de parámetros **Data Import/Export** en el Panel Izquierdo y visualice los parámetros asociados



2. En el grupo de parámetros Solver, realizar las modificaciones solicitadas en el enunciado del ejemplo; y además, desactive la restricción que limita el amacenamiento de las variables tiempo, estados y salida conformada por una cantidad fija de ultimos puntos (observaciones), desactivando la casilla de verificación **Limits data points to last** del grupo de parámetros Data Import/Export.

3. Iniciar la simulación y visualizar las señales de salida resultantes de la simulación

   - Para iniciar la simulación del modelo, elegimos Simulation|Run o damos clic en
   - Luego, para visualizar los resultados damos doble clic al bloque Scope.



4. Visualizamos las variables de salida resultantes de la simulación

   Estas variables son las que hemos configurado para que sean almacenadas en el workspace al finalizar la simulación. Para ello, desde la ventana de comando digitamos el comando whos verificando la presencia de las variables `S_test1_signals` y `tout`

```
>> whos
  Name                    Size              Bytes  Class     Attributes
  S_test1_signals         1x1               49066  struct
  tout                 2001x1               16008  double
```

5. Analicemos los campos que componen la estructura `S_test1_signals`

```
>> S_test1_signals
S_test1_signals =
         time: [2001x1 double]
      signals: [1x1 struct]
    blockName: 's_test1/Scope'
```

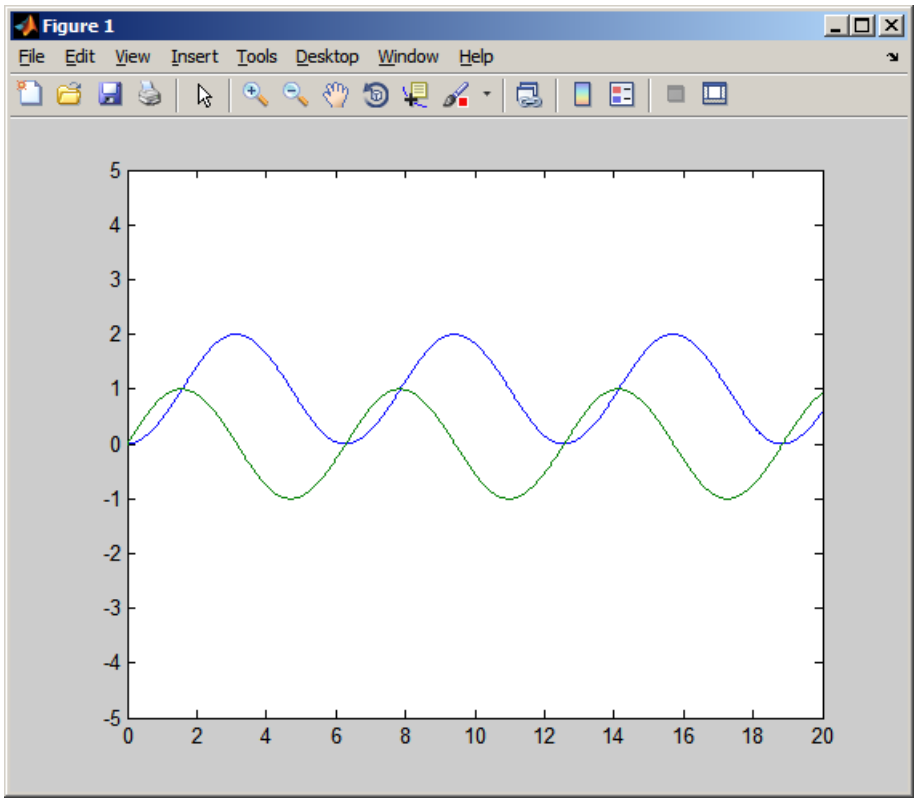
observamos que la variable **time** contiene valores de simulación desde 0 (start time) hasta 20 (stop time) con un paso fijo de 0.01 (fixed-step), razon por la cual éste vector contiene 2001 elementos (observaciones); la variable blockName es una cadena de texto que contiene el nombre del modelo seguido del bloque en el que se haya la salida; y, la variable signals es a su vez otra estructura cuyos campos se muestran a continuación

```
>> S_test1_signals.signals
ans =
        values: [2001x2 double]
    dimensions: 2
         label: ''
         title: ''
     plotStyle: [0 0]
```

6. observamos que la variable values, contiene las dos señales de salida, una por columna. Por lo tanto, podemos también visualizar la gráfica resultando digitando desde la ventana de comandos

```
>> plot(tout,S_test1_signals.signals.values)
>> ylim([-5 5])
```

Obteniendo como resultado



Ejercicios

1. Pruebe el sistema `s_test1` con diferentes tamaños paso de simulación y usando el parámetro **step size**.

   Compare, en particular, el cálculo del tiempo (número de observaciones) para un tamaño de paso de 0.00001, usando ode3 (paso fijo) directamente con step size; y luego, con la respuesta usando ode23 (paso variable).

   Interpretar los resultados. ¿Son muy diferentes las gráficas de las salidas resultantes?

2. ¿Porqué el resultado de la simulación del sistema de prueba s_test1 muestra la solución de la ecuación diferencial con valor inicial

   $$\dot{y}(t) = x(t),\ y(0) = 0\ ?$$

   ¿Cuál de las señales es $x(t)$ y cual $y(t)$?

3. Diseñe el sistema de prueba Simulink `s_soldiff` para el bloque **Derivative**. Para esto es mejor modificar el sistema `s_test1`.

   Luego realice los mismos experimentos que en el problema 1.

4. Piense en como uno podria resolver el problema de valor inicial

   $$\dot{u}(t) = -2u(t),\ u(0) = 1$$

   utilizando Simulink y con la ayuda del bloque **Integrator** y establecer un sistema de Simulink para este problema.

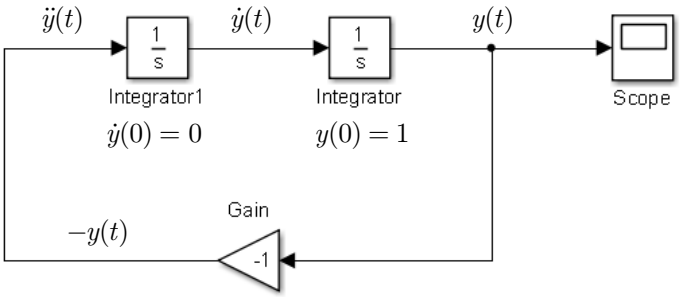
   Compare la solución numérica resultante con la solución exacta.

6.3. Solución de Ecuaciones Diferenciales con Simulink

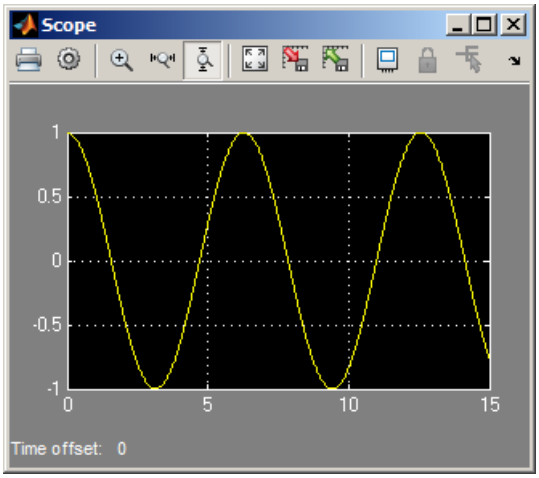
1. Resolver

$$\ddot{y}(t) = -y(t),\ y(0) = 1, \dot{y}(0) = 0$$

- **Sistema Simulink**



- **Salida**

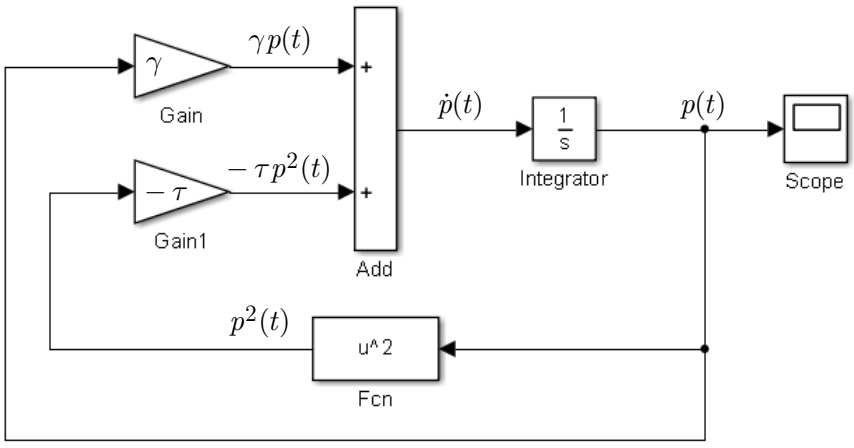


2. Resolver la Ecuación Diferencial Logística

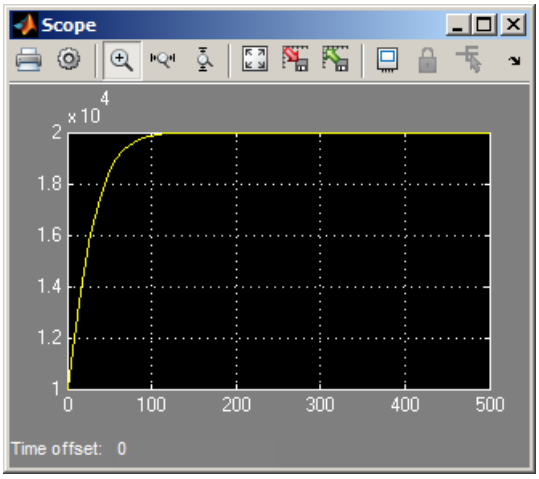
$$\dot{p}(t) = \gamma p(t) - \tau p^2(t)$$

con $\gamma = 0{,}05$, $\tau = 0{,}0000025$,$p(0) = 1000$

- **Sistema Simulink**



- **Salida**

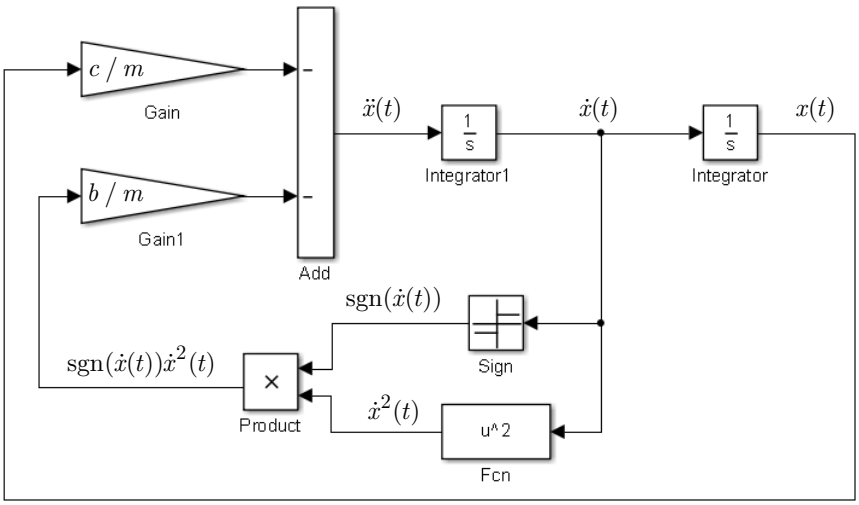


3. Crear un sistema el modelo de oscilaciones mecánica

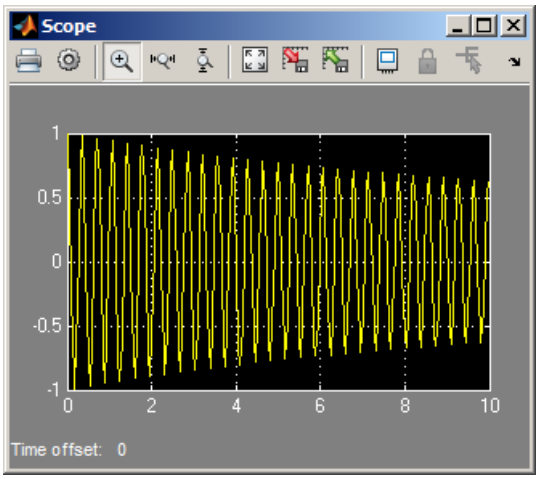
$$m\ddot{x}(t) + b \cdot \text{sign}(\dot{x}(t)) \cdot \dot{x}^2(t) + cx(t) = 0$$

con $m = 0{,}5$ kg, $b = 0{,}00411\frac{\text{kg}}{\text{m}}$, $c = 155{,}2\frac{\text{N}}{\text{m}}$, $x(0) = 1$ m, y $\dot{x}(0) = 0\frac{\text{m}}{\text{s}}$

- **Sistema Simulink**



- Salida

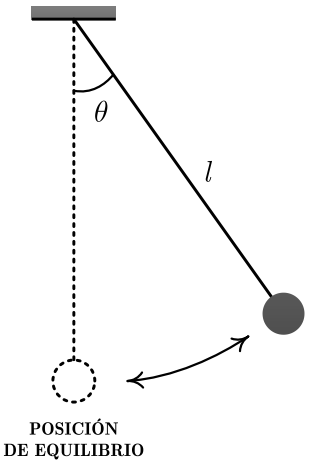


Ejercicios

1. Estamos interesados en la solución de la ecuación diferencial llamada del péndulo matemático

$$\ddot{\theta}(t) = -\frac{g}{l}\text{sen}(\theta(t)), \qquad g = 9{,}81\frac{\text{m}}{\text{s}^2}$$

donde $\theta(t)$ es el angulo que el péndulo (de longitud l) forma en el tiempo relativo a su posición de equilibrio



considerando $l = 10$, $\theta(0) = \frac{\pi}{4}$, $\dot{\theta}(t) = 0$, diseñe y pruebe un sistema Simulink para la solución de la ecuación diferencial del péndulo matemático.

2. Diseñe y pruebe un sistema Simulink para resolver el problema de valor inicial

$$\ddot{y}(t) + y(t) = 0, \quad y(0) = 1, \dot{y}(0) = 0$$

Compara este resultado con la solución exacta. Luego, incorpore la capacidad de simular una perturbación (esto es, una no homogeneidad, introducioendo en el lado derecho de la ecuación diferencial una función $u(t) \neq 0$, denominada función de peturbación) en el sistema Simulink. Obtenga la solución considerando la siguiente función de perturbación $u(t) = e^{-t}$.

3. Resuelva el problema de valor inicial

$$t\ddot{y}(t) + 2\dot{y}(t) + 4y(t) = 4,\ y(1) = 1, \dot{y}(1) = 1$$

mediante un sistema Simulink.

4. Resolver el sistema de ecuaciones diferenciales de primer orden

$$\begin{array}{rcrcrl} \dot{y}_1(t) & = & -3y_1(t) & - & 2y_2(t), & y_1(0) = 1 \\ \dot{y}_2(t) & = & 4y_1(t) & + & 2y_2(t), & y_2(0) = 1 \end{array}$$

mediante un modelo Simulink. Compare la solución numérica obtenida con la solución exacta, la cual puede ser calculada a mano o con la ayuda del Symbolic Math Toolbox.

5. El péndulo matemático doble está conformado por dos péndulos matemáticos simples de longitudes l_1 y l_2 acoplados, de los que cuelgan las masas m_1y m_2 respectivamente. Dado un instante de tiempo t, los hilos inextensibles forman ángulos θ_1 y $\theta_2$con la vertical. El sistema de ecuaciones diferenciales de segundo orden para este sistema es

$$\begin{array}{rcrcrcrcl} l_1\ddot{\theta}_1(t) & + & g\theta_1 & + & l_2\left(\dfrac{m_2}{m_1+m_2}\right)\ddot{\theta}_2 & = & 0 \\ l_2\ddot{\theta}_1(t) & + & g\theta_2 & + & l_1\ddot{\theta}_1 & = & 0 \end{array}$$

6.4. Modelamiento de Sistemas Dinámicos en Simulink en detalle

6.4.1. Semántica de los Diagramas de Bloque

Un modelo clásico de diagrama de bloques de un sistema dinámico consiste gráficamente de bloques y líneas (señales). La historia de estos modelos de diagrama de bloque se deriva de las áreas de ingeniería, tales como **Teoría de Control** y el **Procesamiento de Señales**. Un bloque dentro de un diagrama de bloques define un sistema dinámico en sí mismo. Las relaciones entre cada sistema dinámico elemental en un diagrama de bloques son ilustradas mediante el uso de señales que conectan los bloques. Colectivamente los bloques y líneas en un diagrama de bloques describen un sistema dinámico genérico.

Los productos Simulink extienden estos modelos clásicos de diagrama de bloques introduciendo la noción de dos tipos de bloques, **bloques no virtuales** y **bloques virtuales**. Los **bloques no virtuales** representan sistemas elementales. Los bloques virtuales existen sólo por conveniencia gráfica y de organización: no tienen ningún efecto en el sistema de ecuaciones descritas por el modelo de diagrama de bloques. Puede utilizar bloques virtuales para mejorar la legibilidad de sus modelos.

En general, los bloques y las líneas pueden utilizarse para describir muchos "modelos de cálculos". Un ejemplo sería un diagrama de flujo. Un diagrama de flujo consiste de bloques y líneas, sin embargo, no se pueden describir los sistemas dinámicos genéricos utilizando la semántica de diagramas de flujo.

El término "diagrama de bloques basado en el tiempo" se utiliza para distinguir los diagramas de bloques que describen los sistemas dinámicos de las otras formas de diagramas de bloques, y el término diagrama de bloque (o modelo) es usado para hacer referencia a un diagrama de bloques basada en el tiempo a menos que el contexto requiera una distinción explícita.

Para resumir el significado de los diagramas de bloques basados en el tiempo:

- Los **diagramas de bloques Simulink** definen las relaciones basadas en el tiempo entre las señales y variables de estado. La solución de un diagrama de bloques se obtiene mediante la evaluación de estas relaciones en el tiempo, donde el tiempo se inicia en un "tiempo de inicio" (start time) especificado por el usuario y termina en un "tiempo de finalización" (stop time) especificado por el usuario. Cada evaluación de estas relaciones se conoce como un paso de tiempo (step time).

- Las **señales** representan cantidades que cambian con el tiempo y que se definen para todos los puntos en el tiempo entre el tiempo de inicio y finalización del diagrama de bloques.

- Las relaciones entre las señales y variables de estado se definen por un conjunto de ecuaciones representadas por los bloques. Cada bloque se compone de un conjunto de ecuaciones (métodos de bloque). Estas ecuaciones definen una relación entre las señales de entrada, señales de salida y las variables de estado. Inherente a la definición de una ecuación es la noción de parámetros, que son los coeficientes se encuentran dentro de la ecuación.

6.4.2. Creación de Modelos

Simulink provee un editor gráfico que permite crear y conectar las instancias de tipos de bloques seleccionados de las bibliotecas de tipos de bloque a través de un navegador de bibliotecas (Simulink Library Browser). Loas bibliotecas de bloques son provistas representando sistemas elementales que pueden ser usados como bloques de construcción. Los bloques suministrados con Simulink se denominan bloques built-in. Los usuarios también pueden crear sus propios tipos de bloques y usar el editor de Simulink para crear instancias de ellos en un diagrama. Los bloques definidos por el usuario se denominan bloques personalizados.

6.4.3. Tiempo

El tiempo es un componente inherente de los diagramas de bloques en el que los resultados de una simulación diagrama de bloques cambia con el tiempo. Dicho de otra manera, un diagrama de bloques representa el comportamiento instantáneo de un sistema dinámico. Determinando el comportamiento de un sistema en el tiempo implica resolver repetidamente el modelo en intervalos, llamados pasos de tiempo (time steps), desde el inicio del intervalo de tiempo hasta el final del intervalo de tiempo. El proceso de resolución de un modelo en pasos de tiempo sucesivos se conoce como la *simulación* de el sistema que el modelo representa.

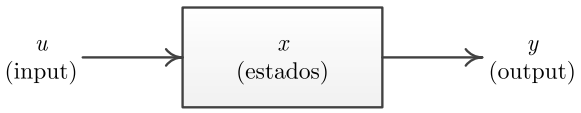
6.4.4. Estados (states)

Son típicamente los valores actuales de algún sistema, y por lo tanto del modelo, las salidas son funciones de los valores previos de las variables temporales. Estas variables se denominan estados (states). El cálculo de las salidas de un modelo a partir de un diagrama de un diagrama de bloques implica guardar el valor de los estados en el paso de tiempo actual para su uso en el cálculo de las salidas en un paso de tiempo posterior. Esta tarea se lleva a cabo durante la simulación para los modelos que definen los estados.

Dos tipos de estados pueden ocurrir en un modelo de Simulink: estados discretos y continuos. Un estado continuo cambias continuamente. Ejemplos de estados continuos son la posición y la velocidad de un automóvil. Un estado discreto es una aproximación de un estado continuo en el que el estado es actualizado (recalculado) usando intervalos finitos (periódicos o no periódicos). Un ejemplo de un estado discreto sería la posición de un vehículo que se muestra en un odómetro digital donde se actualiza cada segundo en lugar de continuamente. En el límite, como el intervalo de tiempo de estado discreto se aproxima a cero, se convierte en un estado discreto equivalente a un estado continuo.

Los **bloques** definen implícitamente los estados de un modelo. En particular, un bloque que necesita algunas o todas sus sus salidas anteriores para calcular sus salidas corrientes (actuales) define implícitamente un conjunto de estados que necesitan ser guardado entre los pasos de tiempo. Este bloque se dice que *tiene estados*.

La siguiente es una representación gráfica de un bloque que *tiene estados*:



Los bloques que definen estados continuos incluyen los siguientes bloques estándar de Simulink:

- Integrator
- State-Space
- Transfer Fcn
- Variable Transport Delay
- Zero-Pole

El número total de estados de un modelo es la suma de todos los estados definidos por todos los bloques. Determinar el número de estados en un diagrama necesita analizar el diagrama para determinar los tipos de bloques que contiene y luego agregar el número de estados definidos por cada instancia de un tipo de bloque que define los estados. Esta tarea se lleva a cabo durante la fase de compilación de una simulación.

Trabajar con los Estados

Las siguientes facilidad se proporcionan para la determinación, inicialización y registro de los estados de un modelo en la simulación:

- El comando **model** muestra información acerca de los estados definidos por un modelo, como el número total de estados definidos por el modelo, el bloque que define cada estado, y el valor inicial de cada estado.
- El depurador de Simulink (debugger) muestra el valor de un estado en cada paso de tiempo durante la simulación, y el comando de depuración **states** de Simulink muestra información sobre los actuales estados de la modelo.
- La importación/exportación de datos de panel de los parámetros de configuración del modelo permite especificar los valores iniciales de los estados de un modelo, y registrar los valores de los estados en cada paso de tiempo durante la simulación, como una variable matriz o estructura en el workspace de MATLAB.
- El cuadro de diálogo Parámetros de bloque (y el paremtro **ContinuousStateAttributes**) permite dar nombres a los estados para los bloques (tales como el integrador) que emplean estados continuos. Esto puede simplificar el análisis de los datos registrados para los estados, especialmente cuando un bloque tiene múltiples estados.

  El modelo de dos cilindros con restricciones de carga ilustra el registro de estados continuos.

Estados Continuos

El cálculo de un estado continuo implica conocer su tasa de cambio, o derivada. Dado que la tasa de cambio de un estado continuo típicamente cambia continuamente por si misma (es decir, está por si misma un estado), el cálculo del valor de un estado continuo en el paso de tiempo actual implica la integración de su derivada desde el inicio de una simulación. Por lo tanto el modelado de un estado continuo implica la representación de la operación de integración y el proceso de cálculo de la derivada de estado en cada punto en el tiempo. Los diagramas de bloques Simulink utilizan bloques Integrator para indicar la integración y una cadena de bloques conectados a la entrada del bloque integrador para representar el método de cálculo de la derivada del estado. La cadena de bloques conectados a la entrada del bloque integrador es la contraparte gráfica de una ecuación diferencial ordinaria (ODE).

En general, con exclusión de los sistemas dinámicos simples, los métodos analíticos no existen para la integración de los estados de los sistemas dinámicos en el mundo real representados por ecuaciones diferenciales ordinarias. La integración de los estados requiere el uso de métodos numéricos llamada solucionadores ODE (ODE solvers). Estos diversos métodos intercambian exactitud de cálculo para cargas de trabajo de cálculo. Simulink cuenta con implementaciones computarizadas de los métodos más comunes de integración ODE y permite que el usuario determine cual será utilizada para integrar los estados representados por bloques Integrator cuando se simula un sistema.

El cálculo del valor de un estado continuo en el paso de tiempo actual implica la integración de sus valores desde el inicio de la simulación. La precisión de la integración numérica a su vez depende del tamaño de los intervalos entre los pasos de tiempo. En general, cuanto menor es el paso de tiempo, más precisa será la simulación. Algunos solucionadores de ODE, llamados solucionadores de paso de tiempo variable (variable time step solvers), puede variar automáticamente el tamaño del paso de tiempo, sobre la base de la tasa de cambio del estado, para conseguir un nivel especifico de exactitud en el transcurso de una simulación. El usuario puede especificar el tamaño del paso de tiempo en el caso de solucionadores de paso fijo (fixed-step solvers), o el solucionador puede determinar automáticamente el tamaño del paso en el caso de solucionadores de paso variable. Para reducir al mínimo la carga de trabajo de cálculo, el solucionador de paso variable elige el tamaño de paso más grande consistente con lograr un nivel general de precisión especificada por el usuario para el estado del modelo que cambia más rápidamente. Esto asegura que todos los estados del modelos se calculan con la precisión especificada por el usuario.

Estados discretos

El cálculo de un estado discreto requiere conocer la relación entre su valor en el intervalo de tiempo actual y su valor en el paso de tiempo anterior. Esto es referido a esta relación como la función de actualización del estado. Un estado discreto no sólo depende de su valor en el paso de tiempo anterior, sino también en los valores de las entradas de un modelo. Modelar un estado discreto implica modelar la dependencia del estado en las entradas de los sistemas en el paso de tiempo anterior. Los diagramas de bloques Simulink utilizan tipos específicos de bloques, llamados bloques discretos, para especificar las funciones de actualización y las cadenas de bloques conectados a las entradas de bloques discretos para modelar la dependencia de los estados discretos de un sistema en sus entradas.

Al igual que con los estados continuos, los estados discretos establecen una restricción en el tamaño de paso de tiempo de simulación. En concreto, el tamaño del paso debe garantizar que todos los tiempos muestrales (sample times) de los estados de los estados del modelo son hit. Esta tarea se asigna a un componente del sistema de Simulink llamado un programa de solución discreta (discrete solver). Se proporcionan dos solvers discretos: un solucionador discreto de paso fijo (fixed-step discrete solver) y un solucionador discreto de paso variable (variable-step discrete solver). El solucionador discreto de paso fijo determina un tamaño de paso fijo que golpea todos los tiempos de la muestra de todos los estados discretos del modelo, independientemente de si los estados realmente cambian de valor en el que golpea el tiempo muestral. Por el contrario, el solucionador discreto de paso variable varía el tamaño del paso para asegurarse de que el sample time hits se produzca sólo en los momentos cuando los estados cambian de valor.

Modelado de Sistemas Híbridos

Un sistema híbrido es un sistema que tiene ambos estados, discretos y continuos. Estrictamente hablando, cualquier modelo que tiene tiempos de muestreo (sample time), tanto continuo como discreto, es tratado como un modelo híbrido, suponiendo que el modelo tiene ambos estados, continuos y discretos. La solución de tal modelo implica la elección de un tamaño de paso que satisfaga tanto la restricción de

precisión de la integración en estado continuo y el sample time hit restringido sobre los estados discretos. El software Simulink satisface este requisito al pasar el siguiente sample time hit, tal como es determinado por el solucionador discreto, como una restricción adicional en el solucionador continuo. El solucionador continuo debe elegir un tamaño de paso que avance la simulación pero no más allá de timepo del siguiente sample time hit. El solucionador continuo puede tomar un paso de tiempo corto del proximo sample time hit para satisfacer su restricción de precisión pero no puede timar un paso más allá del sample time hit ncluso si su restricción de precisión lo permite.

Se pueden simular sistemas híbridos utilizando cualquiera de los métodos de integración, pero algunos métodos son más eficaces que otros. Para la mayoría de sistemas híbridos, ode23 y ode45 son superiores a los otros solucionadores en términos de eficiencia. Debido a las discontinuidades asociadas con el muestreo y retención de los bloques discretos, no es convenietne usar los solucionadores ode15s y ode113 para sistemas híbridos.

6.4.5. Los Parámetros de Bloque

Las propiedades clave de muchos bloques estándar están parametrizadas. Por ejemplo, el valor constante de la bloque Constant de Simulink es un parámetro. Cada bloque parametrizado tiene un bloque de diálogo que le permite establecer los valores de los parámetros. Puede utilizar expresiones de MATLAB para especificar los valores de los parámetros. Simulink evalúa las expresiones antes de ejecutar una simulación. Puede cambiar los valores de los parámetros durante la simulación. Esto le permite determinar interactivamente el valor más adecuado para un parámetro.

Un bloque de parámetros representa efectivamente una familia de bloques similares. Por ejemplo, al crear un modelo, puede establecer el parámetro de valor constante de cada instancia del bloque Constant por separado para que cada instancia se comporte de manera diferente. Debido a que esto permite que cada bloque estándar represente una familia de bloques, la parametrización de bloques aumenta en gran medida el poder de modelado de las bibliotecas de Simulink estándar.

6.4.6. Parámetros ajustables

Muchos de los parámetros de bloque son ajustables. Un *parámetro ajustable* es un parámetro cuyo valor puede cambiar sin tener que recompilar el modelo. Por ejemplo, el parámetro del bloque Gain es ajustable. Puede modificar la ganancia del bloque, mientras que la simulación se ejecuta. Si un parámetro no se puede ajustar y se ejecuta la simulación, el cuadro de diálogo de control que establece el parámetro está desactivado.

Cuando se cambia el valor de un parámetro ajustable, el cambio toma efecto al inicio del próximo paso de tiempo.

6.4.7. El Bloque de Tiempos Muestrales

Cada bloque de Simulink tiene un tiempo de muestreo (sample time) que define cuando el bloque se ejecutará. La mayoría de los bloques permiten especificar el tiempo de muestreo través del parametro **SampleTime**. Las opciones comunes incluyen, tiempos de muestra continuos, discretos y heredados (inherited).

Tipos de Tiempo de Muestreo Común	**Tiempo de Muestreo**	**Ejemplos**
Discreto	$[T_s, T_0]$	Unit Delay, Digital Filter
Contínuo	$[0, 0]$	Integrator, Derivative
Heredado	$[-1, 0]$	Gain, Sum

Para bloques discretos, el tiempo de muestreo es un vector $[T_s, T_0]$, donde T_s es el intervalo de tiempo o período entre tiempos de muestreo consecutivos y T_0 es un desplazamiento inicial al tiempo de muestreo. En contraste, los tiempos de muestreo para los bloques no discretos están representados por pares ordenados que utilizan cero, un número entero negativo, o infinito para representar un tipo específico de tiempo de mustreo. Por ejemplo, los bloques Continuous tienen un tiempo de muestreo nominal de [0, 0] y son utilizados para modelar sistemas en los cuales los estados cambian continuamente (por ejemplo, la aceleración de un auto). Mientras que el tipo de tiempo de muestreo de un bloque heredado simbólicamente como [-1, 0] y Simulink a continuación determina el valor actual basado en el contexto del bloque heredado dentro del modelo.

Se debe tener en cuenta que no todos los bloques aceptan todos los tipos de tiempos de muestreo. Por ejemplo, un bloque discreto no puede aceptar un tiempo de muestreo continuo.

Para una ayuda visual, Simulink permite la opcional codificación por color y la anotación de cualquier diagrama de bloques para indicar el tipo y la velocidad de los tiempos de muestreo del bloque. Se puede capturar todos los colores y las anotaciones en una leyenda.

6.4.8. Bloques personalizados

Se pueden crear bibliotecas de bloques personalizadas que se pueden utilizar en los modelos. Se puede crear un bloque personalizado ya sea gráficamente o mediante programación. Para crear un bloque personalizado gráficamente, se dibuja un diagrama de bloques que representa el comportamiento del bloque, se envuelve este diagrama en una instancia del bloque de subsistema Simulink, y se proporciona el bloque con un cuadro de diálogo de parámetros, mediante el uso de las máscaras de bloques Simulink. Para crear un bloque de programación, se crea un archivo de MATLAB o un archivo MEX que contenga las funciones del sistema del bloque. El archivo resultante se denomina función S. Luego se asocia la función S con las instancias del bloque S-Function de Simulink en el modelo. Se puede agregar un cuadro de diálogo de parámetros al bloque S-Function envolviéndolo en un bloque de subsistema y añadiendo el cuadro de diálogo de parámetros al bloque subsistema.

6.4.9. Sistemas y subsistemas

Un diagrama de bloques Simulink puede consistir de capas. Cada capa está definida por un subsistema. Un subsistema es parte del diagrama general de bloques e idealmente no tiene ningún impacto sobre el significado del diagrama de bloques. Los subsistemas son provistos principalmente para ayudar con los aspectos organizativos de un diagrama de bloques. Los subsistemas no definen un diagrama de bloques separado.

El software Simulink distingue entre dos tipos diferentes de subsistemas: virtuales y no virtuales. La principal diferencia es que los subsistemas no virtuales proporcionan la capacidad de controlar cuando se evalúan los contenidos del subsistema.

Los subsistemas virtuales

Los subsistemas virtuales proporcionan jerarquía gráfica en los modelos. Los subsistemas virtuales no afectan la ejecución. Durante la ejecución del modelo, el motor de Simulink aplana todos los subsistemas virtuales, es decir, Simulink expande el subsistema en su lugar antes de la ejecución. Esta expansión es muy similar a la forma en que las que trabajan las macros en un lenguaje de programación como C o C++. En términos generales, habrá un sistema para el diagrama de bloques de nivel superior que se conoce como el sistema raíz, y varios sistemas de nivel inferior derivados de subsistemas no virtuales y otros elementos en el diagrama de bloques. Estos sistemas son vistos en el depurador de Simulink. El acto de crea estos sistemas internos se conoce a menudo como el *aplanamiento de la jerarquía del modelo.*

Los subsistemas no virtuales

Los subsistemas no virtuales, que se dibujan con un borde en negrita, proporcionan la ejecución y la jerarquía gráfica en los modelos. Los subsistemas no virtuales se ejecutan como una sola unidad (ejecución atómica) por el motor de Simulink. Puede crear subsistemas ejecutados condicionalmente, que se ejecutan sólo cuando una condición previa, como un gatillador (trigger), una llamda a funcón, o una acción ocurre. Simulink siempre calcula todos los insumos utilizados durante la ejecución de un subsistema no virtual antes de ejecutar el subsistema. Simulink define los siguientes subsistemas no virtuales.

Los Subsistemas Atómicos. La principal característica de un subsistema atómico es que los bloques en un subsistema atómico se ejecutan como una sola unidad. Esto proporciona la ventaja de agrupar los aspectos funcionales de los modelos en el nivel de ejecución. Cualquier bloque de Simulink se puede colocar en un subsistema atómico, incluyendo bloques con diferentes tasas de ejecución. Se puede crear un subsistema atómico seleccionando la opción Treat as atomic unit (tratar como unidad atómica) en un subsistema virtual.

Los Subsistemas Habilitados. Un subsistema habilitado se comporta de manera similar a un subsistema atómica, excepto que sólo se ejecuta cuando la señal de activación del puerto de habilitación del subsistema es mayor que cero. Para crear un subsistema habilitado, se coloca un bloque activador de puerto dentro de un bloque de subsistema. Se puede configurar un subsistema habilitado para mantener o restablecer los estados de los bloques dentro de la acción antes de la acción habilitante del subsistema. Sólo se tiene que seleccionar el parámetro **States when enabling** del bloque activar puerto. Del mismo modo, se puede configurar cada puerto de salida de un subsistema habilitado para mantener o restablecer su salida antes de la acción deshabilitantes del subsistema. Se debe seleccionar el parámetro Output when disables (salida cuando se desabilita) en el bloque Outport.

Los Subsistemas Gatillados (Triggered). Se crea un subsistema gatillado mediante la colocación de un bloque de puerto de gatillo dentro de un subsistema. El subsistema resultante se ejecuta cuando un flanco ascendente o descendente con respecto al cero es visto sobre la señal de activación del puerto de gatillado del subsistema. La dirección del borde gatillante se define por el parametro Trigger type en el puerto gaitllante del bloque. Simulink limita el tipo de bloques colocados en un subsistema gatillante a los bloques que no tienen tiempos de muestreo explícitos (es decir, los bloques dentro del subsistema deben tener un tiempo de muestreo de -1) porque los contenidos de un subsistema gatillado se ejecutan de forma aperiódica. Un cuadro Stateflow también puede tener un puerto de gatillado, que se define mediante el editor de Stateflow. Simulink no distingue entre un subsistema gatillado y un cuadro gatillado.

Los Subsistemas de llamada a función. Un subsistema de llamada a función es un subsistema que otro bloque puede invocar directamente durante una simulación. Esto es análogo a una función en un lenguaje de programación procedimental. La invocación de una llamada a funcisón del subsistema es equivalente a invocar la salida y actualizar los métodos de los bloques que el subsistema contiene de forma ordenada. El bloque que invoca un subsistema de función-llamada se denomina un iniciador de la llamada a función. Stateflow, el generador de funciones de llamada, y los bloques de función S se pueden todos serviro como iniciadires de llamadas a función. Para crear un subsistema de llamada a función, arrastre un bloque Subsistema Function-Call (Función-Llamada) de la biblioteca Ports & Subsystems en el modelo y conectar un iniciador de llamada a función en el puerto de llamada a función que aparece en la parte superior del subsistema. También puede crear un subsistema de llamada a Función desde cero, creando primero un bloque de subsistema en su modelo y luego creando un bloque gatillador en el subsistema y fijando el Trigger type del bloque gatillador a function-call.

Se puede configurar un subsistema de la función de llamada que se activará (por defecto) o periódica mediante el establecimiento de su tipode tiempo de muestreo a ser gatillado o periódico, respectivamente. Un iniciador del subsistena llamada a función puede invocar un zero, uno, o multiples veces de pasos por paso de tiempo. Un indicador llamada a función puede invocars un subsistema de llamda a función zero, uno, o muchas veces por paso de tiempo. Los tiempos muestrles de todos los bloques en un subsistema de llamada a función gatillado deben establecerse heredado (-1).

Un iniciador de llamada a función puede invocar un subsistema de llamada a función solo una vez por paso de tiempo y debe invocar el subsistema periódicamente. Si el iniciador invoca un subsistema de llamada a función periódica aperiodicamente, Simulink detiene la simulación y muestra un mensaje de error. Los bloques de un subsistema de llamada a función periódica puede especificar un tiempo de muestreo no heredada o heredada (-1). Todos los bloques que especifican un tiempo de muestreo no heredado deben especificar el mismo tiempo de muestreo, es decir, si un bloque especifica 0.1 como su tiempo de muestreo, todos los otros bloques deben especificar un tiempo de muestreo de 0.1 o -1. Si un iniciador de llamada a función invoca un subsistema de llamada a función periódica a una velocidad que difiere del tiempo de muestreo especificado por los bloques en el subsistema, Simulink detiene la simulación y muestra un mensaje de error.

Los Subsistemas Habilitados y Gatillados. Se puede crear un subsistema habilitado y gatillado mediante la colocación de un bloque Trigger Port y un bloque Enable Port dentro de un bloque de subsistema. El subsistema resultante es esencialmente un subsistema gatillado que se ejecuta cuando el subsistema es habilitado y un flanco ascendente o descendente con respecto a cero ha sido visto en la señal de accionamiento del Trigger Port del subsistema. La dirección del borde de gatillamiento se define por el parámetro Trigger type en el bloque Trigger Port. Debido a que los contenidos de un subsistema habilitado se ejecutan de una manera aperiódica, Simulink limita los tipos de bloques colocados en un subsistema habilitado y gatillado a los bloques que no tienen tiempos de muestreo explícito. En otras palabras, los bloques dentro del subsistema debe tener un tiempo de muestreo de -1 .

Los Subsistemas de Acción. Los Subsistemas de acción pueden ser considerados como una intersección de las propiedades de los subsistemas habilitados y los subsistemas de llamada a función. Los subsistemas de acción están restringidos a un solo tiempo de muestro (por ejemplo, un tiempo de muestreo continuo, discreto, o heredado). Los subsistemas de acción deben ser ejecutados por un iniciador del subsistema de acción. Este puede ser un bloque **If** o un bloque **Switch Case**. Todos los subsistemas de acción conectados a un iniciador de subsistema de acción deben tener el mismo tiempo de muestreo. Un subsistema de acción se crea mediante la colocación de un bloque Action port dentro de un bloque del subsistema. El icono de subsistema se adapta automáticamente al tipo de bloque (es decir, bloques If o Swith Case) que se está ejecutando el subsistema de acción.

Los subsistemas de acción se puede ejecutar como máximo una vez por el iniciador subsistema de acción. Los subsistemas de acción permiten controlar cuando los se reestablece los estados mediante **States when execution is resumen** en el bloque Action Port. Los subsistemas de acción también dan el control sobre si mantener o no los valores de los puertos de salida via el parámetro **Output when disabled** en el bloque outport. Esto es análogo a los subsistemas habilitados.

Los subsistemas de acción se comportan de manera muy similar a los subsistemas de llamada a función, ya que deben ser ejecutados por un bloque iniciador. Los subsistemas llamada a función se pueden ejecutar más de una vez en cualquier intervalo de tiempo determinado, mientras que los subsistemas de acción se puede ejecutar como máximo una vez . Esta restricción significa que un mayor número de bloques (por ejemplo, bloques periódicos) se puede colocar en los subsistemas de acción en comparación con los subsistemas de llamada a función. Esta restricción también significa que se puede controlar la forma en que se comportan los estados y salidas.

Los Subsistemas Iteradores While El subsistema iterador *while* ejecutará varias iteraciones en cada paso de tiempo del modelo. El número de iteraciones es controlado por la condición de **bloque While Iterator**. Un subsistema iterador mientras es creado mediante la colocación de un bloque de iterador Whileque dentro de un bloque del subsistema.

Un subsistema iterador While que es muy similar a un subsistema de llamada de función en que puede ejecutarse para cualquier número de iteraciones a un paso de tiempo dado. El subsistema iterador While que difiere de un subsistema de llamada a función en que no hay un iniciador por separado (por ejemplo, un gráfico de flujo de estados). Además, un subsistema iterador While tiene acceso al número de iteraciones actual opcionalmente producido por el bloque iterador While. Un subsistema iterador While también provee control control sobre si debe o no restablecer los estados cuando se inicia a través del parámetro **States when starting** en bloque iterador While.

Los Subsistemas Iteradores For El subsistema iterador *for* se ejecutará un número fijo de iteraciones en cada paso de tiempo del modelo. El número de iteraciones puede ser una entrada externa a la para el subsistema del iterador o especificado internamente en el bloque Para iterador. Un iterador para el subsistema se crea mediante la colocación de un bloque de iteradores Para dentro de un bloque de subsistema.

Un iterador For tiene acceso al número de iteración corriente que es opcionalmente producido por el bloque iterador For. Un subsistema iterador For también da control sobre si debe o no restablecer los estados cuando se inicia a través de el parámetro States when starting el bloque iterador For. Un subsistema iterador For es muy similar a un subsistema iterador While con la restricción de que se fija el número de iteraciones durante cualquier intervalo de tiempo dado.

Los Subsistemas Iteradores For Each El subsistema iterador *for each* permite repetir un algoritmo para elementos individuales (o subarreglos) de una señal de entrada. Aquí, el algoritmo está representado por el conjunto de bloques en el subsistema y se aplica a un solo elemento (o subarreglo) de la señal. Se puede configurar la descomposición de las entradas del subsistema en elementos (o subarreglos) utilizando el bloque For Each, que reside en el subsistema. El bloque For Each también permite configurar la concatenación de los resultados individuales en las señales de salida. Una ventaja de este subsistema es que mantiene separados los conjuntos de estados para cada elemento o subconjunto que esté procesando. Además, en ciertos modelos, el subsistema el for de cada subsistema mejora la reutilización de código del código generado por Simulink Coder.

6.4.10. Las señales

El término *señal* se refiere a una cantidad variable en el tiempo que tiene valores en todos los puntos en el tiempo. Puede especificar una amplia gama de atributos de señales, incluyendo el nombre de la señal, el tipo de dato (por ejemplo, enteros de 8 bits, 16 bits o 32 bits), tipo numérico (real o complejo), y dimensionalidad (unidimensional, dos-dimensional, o una matriz multidimensional). Muchos bloques pueden aceptar señales de salida de cualquier dato o tipo numérico y con dimensionalidad. Otros imponen restricciones sobre los atributos de las señales que pueden manejar.

En el diagrama de bloques, las señales están representadas con líneas que tienen una punta de flecha. La fuente de la señal se corresponde con el bloque que escribe en la señal durante la evaluación de sus métodos de bloque (ecuaciones). Los destinos de la señal son bloques que leen la señal durante la evaluación de los métodos del bloque (ecuaciones).

Una buena manera de entender la definición de una señal es considerar un aula. El profesor es el responsable de escribir en la pizarra y los alumnos que lean lo que está escrito en la pizarra ellos lo cuando eligen. Esto también es cierto con las señales de Simulink: un lector de la señal (un método de bloques) se puede elegir la lectura de la señal como frecuente o infrecuente tal como así lo desea.

6.4.11. Los métodos de bloque

Los bloques representan múltiples ecuaciones. Estas ecuaciones se representan como métodos de bloque. Estos métodos de bloque son evaluados (ejecutados) durante la ejecución de un diagrama de bloques. La evaluación de estos métodos de bloque se lleva a cabo dentro de un bucle de simulación, en donde cada ciclo a través del bucle de simulación representa la evaluación del diagrama de bloques en un punto dado en el tiempo.

Tipos de métodos

Los nombres se asignan a los tipos de funciones realizados por los métodos de bloque. Los tipos de métodos comunes incluyen:

- **Outputs,** calculan las salidas de un bloque dadas sus entradas en el paso de tiempo actual y sus estados discretos en el paso de tiempo anterior.
- **Update,** calculan el valor de los estados discretos del bloque en el paso de tiempo actual, dadas sus entradas en el paso de tiempo actual y sus estados discretos en el paso de tiempo anterior.
- **Derivatives,** calculan los derivados de estados continuos del bloque en el paso de tiempo actual, dadas las entradas de los bloques y los valores de los estados en el paso de tiempo anterior.

Convención de nomenclatura

Los método de bloques realizan el mismo tipo de operaciones en diferentes formas para diferentes tipos de bloques. La interfaz de usuario de Simulink y la documentación utiliza la notación de punto para indicar la función específica realizada por un método de bloque:

```
TipoBloque.TipoMetodo
```

Por ejemplo, el método que calcula las salidas de un bloque Gain es referido como

```
Gain.Outputs
```

El depurador de Simulink toma la convención de nombres más allá y utiliza el nombre de instancia de un bloque para especificar tanto el tipo de procedimiento y la instancia del bloque en el que se invoca el método durante la simulación, por ejemplo,

```
g1.Outputs
```

6.4.12. Los métodos del modelo

Además de los métodos de bloque, se proporciona un conjunto de métodos que calculan las propiedades del modelo y sus resultados. El software Simulink invoca igualmente estos métodos durante la simulación para determinar las propiedades de un modelo y sus resultados. Los métodos del modelo generalmente realizan sus tareas mediante la invocación de métodos de bloques del mismo tipo. Por ejemplo, el método Outputs del modelo invoca los métodos de salida de los bloques que contienen en el orden especificado por el modelo para calcular sus salidas. El método Derivatives invoca de manera similar los métodos derivados de los bloques que lo contiene para determinar los derivados de sus estados.

Capítulo 7

Introducción a GUIDE

7.1. La Interfaz Gráfica de Usuario

La interfaz gráfica de usuario, conocida también como GUI (del inglés Graphical User Interface) es un tipo de interfaz de usuario que utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Como en una GUI las acciones se realizan mediante manipulación directa, el usuario no tiene que crear un script, digitar algún comando en la línea de comandos o comprender los detalles de cómo se realizan las tareas para poder hacer alguna actividad con la aplicación. Las GUIs surgen como evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico.

7.1.1. Orígenes de las GUI

Los investigadores del Stanford Research Institute liderados por Douglas Engelbart (Universidad de Berkeley), desarrollaron en 1973 el Xerox Alto, el primer ordenador personal con una interfaz de hipervínculos en modo texto gobernado por un mouse, que también inventaron (el primer prototipo en madera). Este concepto fue ampliado y trasladado al entorno gráfico por los investigadores del Xerox PARC (Palo Alto Research Center); en él se definieron los conceptos de ventanas, checkbox, botones de radio, menús y puntero del mouse. Fue implementado comercialmente en el Xerox Star 8010 en 1981.

Hoy en día, tenemos como ejemplo de GUIs:

- Los entornos de escritorio de los sistemas operativos: Windows, Mac Os, X – Windows (Linux), etc.
- Los entornos que usan sistemas operativos de tiempo real: cajeros automáticos, procesos industriales, teléfonos móviles, etc.

7.2. Las GUIs en MATLAB

Desde el punto de vista de la programación en MATLAB, una GUI es una visualización gráfica de una o mas ventanas que contienen controles, llamados componentes, que permiten a un usuario realizar tareas en forma interactiva.

7.2.1. Los componentes

Los componentes de una GUI en MATLAB son:

7.3. Creación de GUIs con MATLAB

Una GUI MATLAB es una ventana figura (figure) en la cual se añaden los controles operados por el usuario (componentes) . A través de devoluciones de llamada (callbacks) se puede hacer que los componentes hagan lo que se desea cuando el usuario le da clic o los manipula con pulsaciones del teclado (keystrokes).

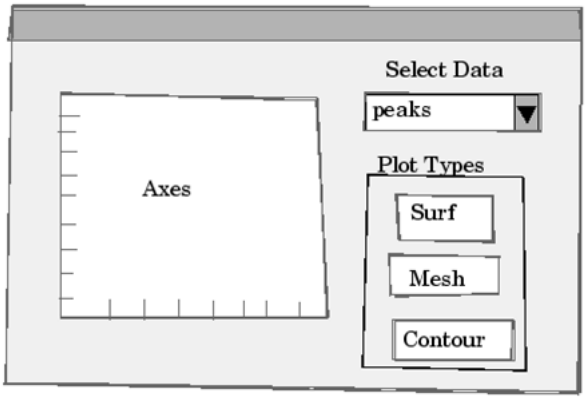
Se puede crear una GUI en MATLAB de dos maneras

1. **Usando GUIDE (GUI Development Environment)**. GUIDE es una herramienta interactiva para la construcción de GUIs Se inicia con una ventana figura en la cual se colocan los componentes desde un editor un editor de diseño. GUIDE crea los archivos M asociados que contienen las devoluciones de llamada (callbacks) para le GUI y sus componentes. GUIDE trabaja con dos tipos de archivo : Archivo para almacenar el diseño de la ventana figura (archivo .fig) Archivo para almacenar el código fuente de la aplicación (archivo .m)

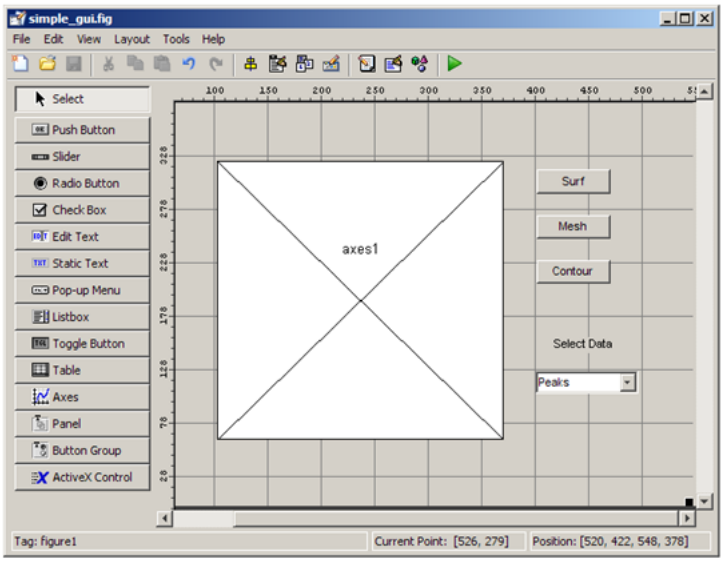
2. **Usando solo archivos M (funciones o script) que generen los GUIs o construcción programática de GUIs.** Aquí, se codifica un archivo M que define todas las propiedades y comportamientos de los componentes; cuando un usuario ejecuta el arcihvo M, se crea una ventana figura con los componentes y los manipuladores interactivos para el usuario.

7.4. Creación de una aplicación GUI con GUIDE

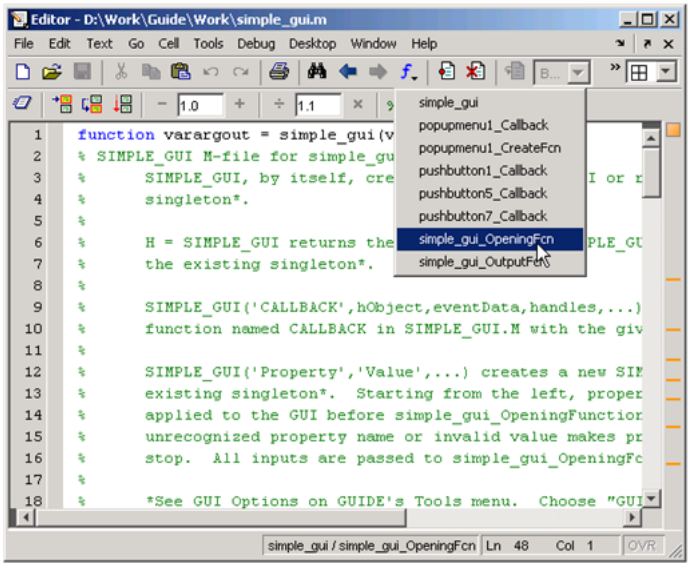
1. Tras un análisis del problema, se propone un esbozo a papel y lápiz de la aplicación GUI.



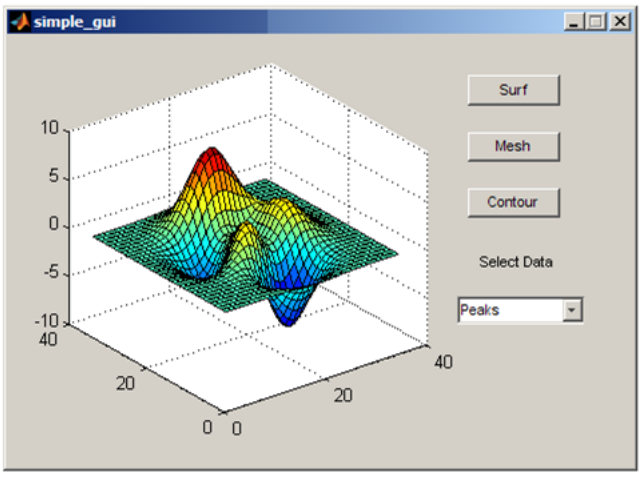
2. Se diseña la GUI colocando los componentes según el esbozo inicial, dándole el aspecto necesario.



3. Se codifica las respuestas a los eventos desencadenados sobre los componentes; es decir, se establece el comportamiento de la aplicación.



4. Se ejecuta la aplicación

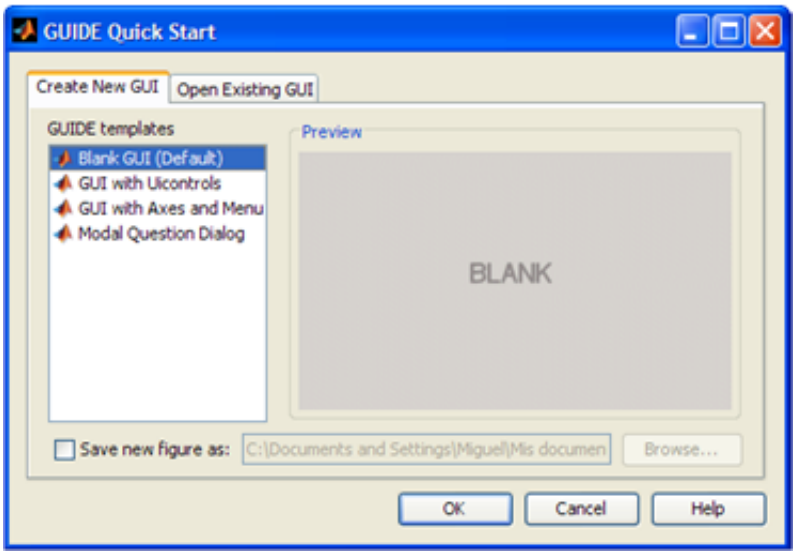


7.5. Estructura de una aplicación GUIDE

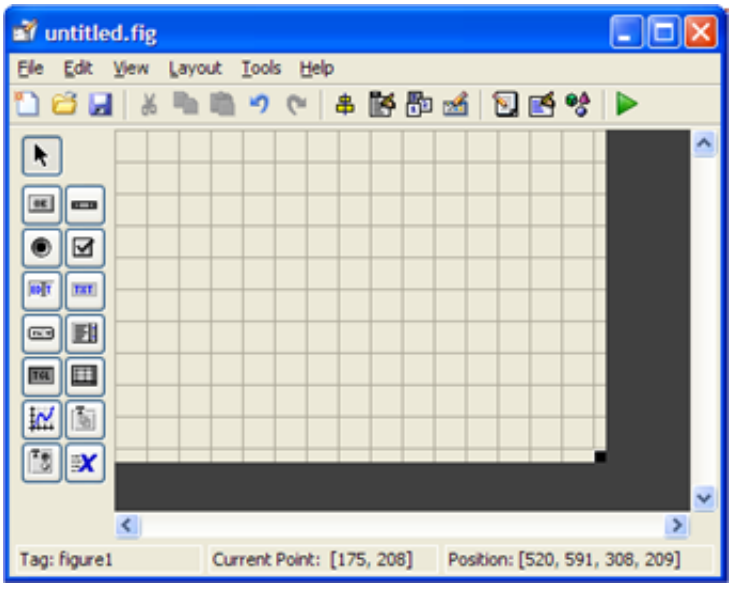
Para iniciar GUIDE digitamos desde la línea de comandos:

```
>> guide
```

se cargará inmediatamente el GUIDE Quick Start.



Elegimos una plantilla (Blank GUI por defecto) y se cargará el GUIDE Layout Editor



7.5.1. Archivos de una aplicación GUIDE

Toda aplicación GUIDE consta como mínimo de dos archivos:

- **Archivo .fig**

  Es un archivo binario que contiene una descripción completa del diseño de la GUI y sus componentes. Este archivo es del tipo .mat, por lo tanto solo se puede modificar en Layout Editor de GUIDE.

- **Archivo .m**

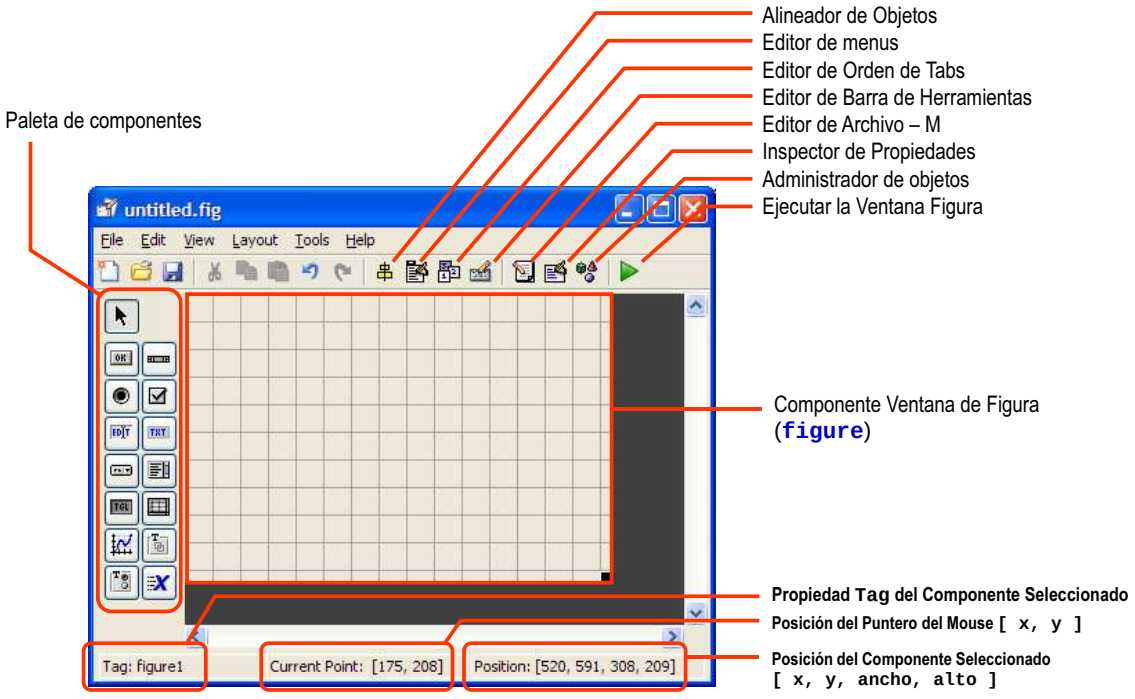
  Es un archivo de texto que contiene código de inicialización y plantillas para la codificación de funciones callback que controlan el comportamiento de la GUI. Dado que éste archivo esta constituido de funciones, el archivo M de la aplicación GUI nunca puede ser un script.

Observaciones:

- Los archivos .fig y .m deben tener el mismo nombre y generalmente deben residir en el mismo directorio. Cuando se guarda por primera vez la aplicación, GUIDE abrirá automáticamente el archivo M en el MATLAB editor.

- Una aplicación GUIDE puede hacer uso de funciones de terceros, por ejemplo, de aquellas que implementan algoritmos numéricos.

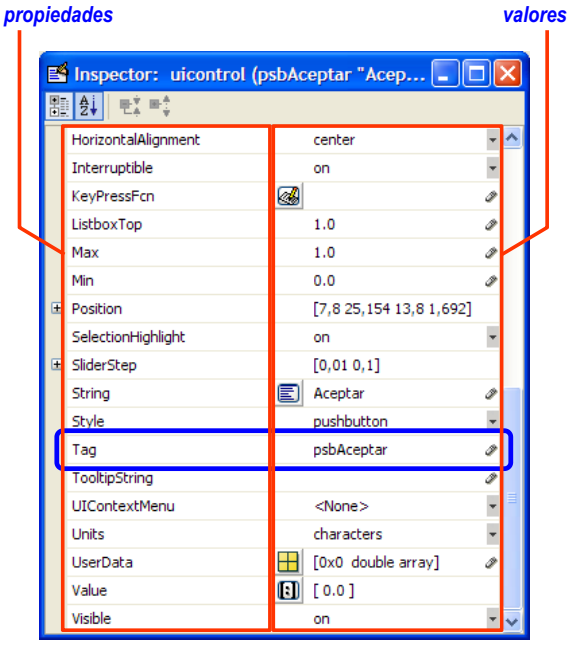
7.6. El GUIDE Layout Editor



7.7. Las Propiedades de los Componentes

Cada componente de la aplicación posee propiedades. Las propiedades permiten establecer las características del componente. Éstas se establecen modificando sus valores. GUIDE provee de la herramienta Property Inspector, para poder realizar el mantenimiento de las propiedades de cada componente.

Cuando se da doble clic en un componente se visualiza el **Propery Inspector** listando las propiedades del componente.



Property Inspector

Cada vez que se añada un nuevo componente a la aplicación, GUIDE asignará valores por defecto a sus propiedades, los cuales podrán ser luego personalizables según la necesidad de la aplicación que estemos desarrollando.

Todos los componentes tienen la propiedad **Tag**, la cual permite referenciar al componente dentro del código fuente. Todos los Tags de una aplicación GUIDE conforman una estructura llamada handles, a través de la cual se hace referencia al componente en el código fuente.

7.8. Estructura del archivo M de una GUI

1. **La Función Principal (de inicialización)**

   El archivo M generado por GUIDE es un archivo del tipo función. El nombre de la función principal es el mismo nombre del archivo M. Aquí se especifican las tareas de inicialización de GUIDE.

   Advertencia: NO se debe editar éste código.

2. **La Función de Apertura (OpeningFcn)**

   Es aquella subfunción que realiza las tareas de inicialización antes de que el usuario tenga acceso a la GUI.

3. **La Función de Salida (OutputFcn)**

   Es aquella subfunción que retorna salidas a la línea de comando MATLAB después de que la función de apertura retorna el control y antes de que el control retorne a la línea de comandos.

4. **Las Funciones callback**

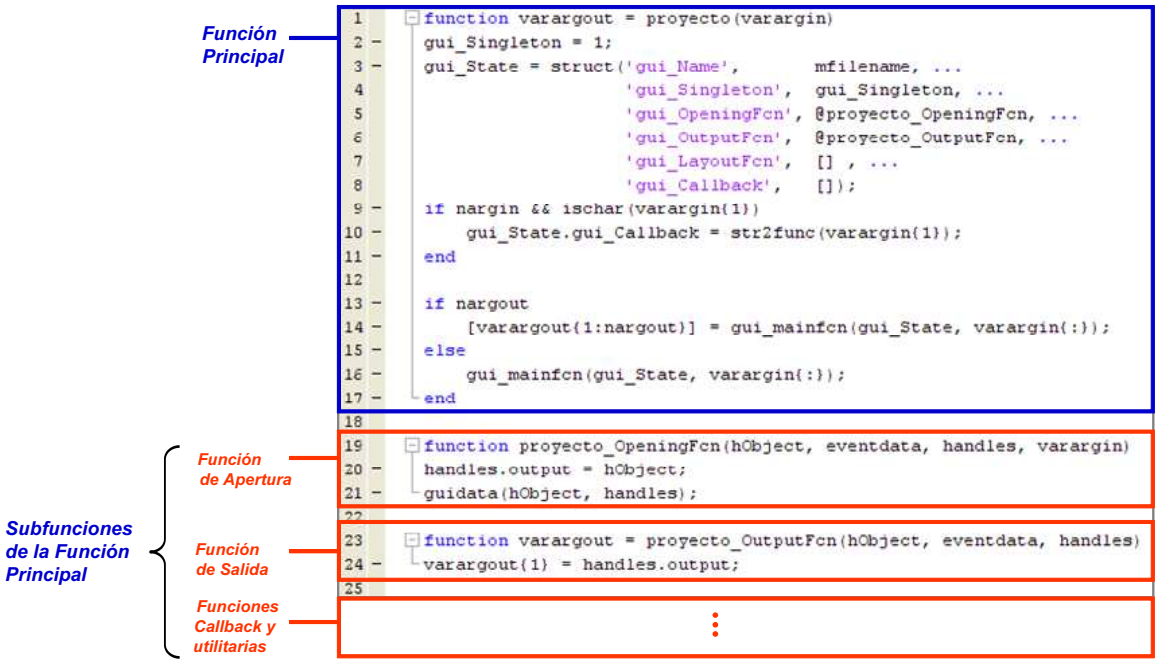
   Cada callback es una subfunción de la función principal. Cuando GUIDE genera un archivo M, éste automáticamente incluye plantillas para las funciones callback de uso frecuente para cada componente.. Los callbacks de los componentes y de la ventana figura, controlan el comportamiento de la Ventana Figura y de los componentes individuales. MATLAB invoca a un callback en respuesta a un evento particular de un componente.

5. **Las Funciones utilitarias o de ayuda**

   Son subfunciones que realizan tareas misceláneas no directamente asociado con algún evento de la ventana figura o de un componente

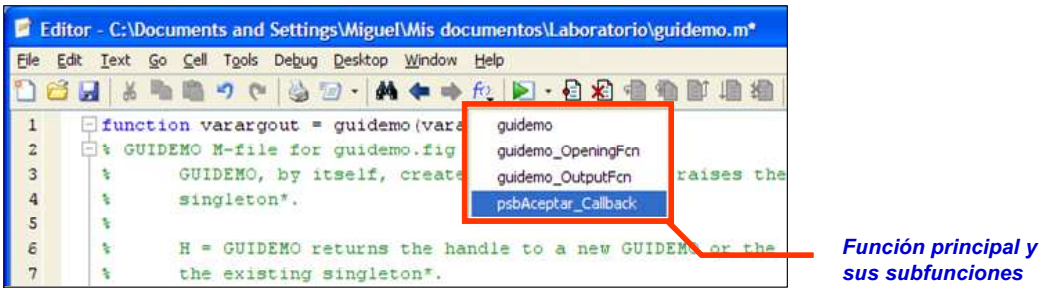
6. **Los Comentarios**

   Son predefinidos por GUIDE o establecidos por el programador.



7.9. Estilo de Programación en GUIDE

Después de haber diseñado la aplicación GUI, se requiere programar su comportamiento; es decir, codificar las respuestas (funciones callback) ante eventos ocurridos sobre alguno de sus componentes. Esto se realiza en el archivo M – función asociado a la aplicación GUIDE

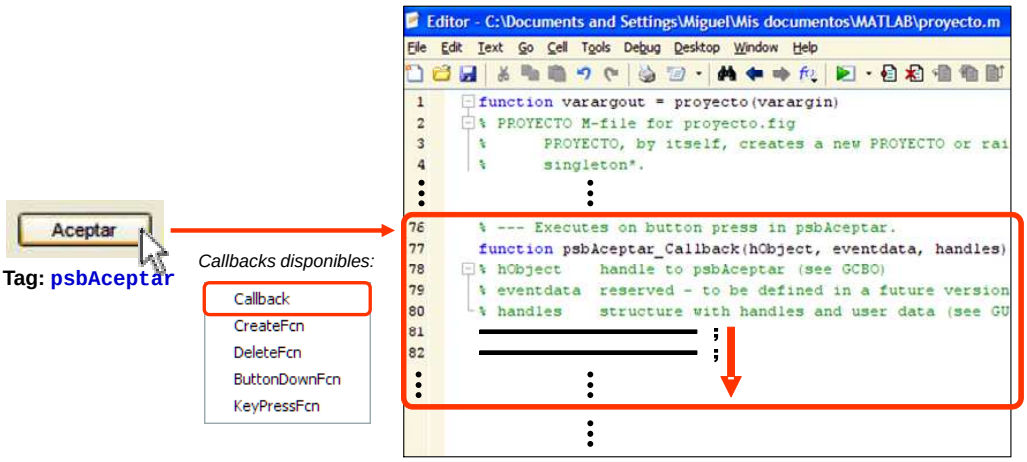


El estilo de programación en GUIDE es estructurado, orientado a componentes y conducido por eventos desencadenados sobre algún componente de la aplicación (funciones callback).|

7.10. Los Callbacks

Un callback, es una subfunción de la función principal de la aplicación, que se ejecuta como respuesta ante un evento desencadenado sobre un componente.

Por ejemplo, un callback muy frecuente es aquel que responde al evento clic izquierdo del mouse.

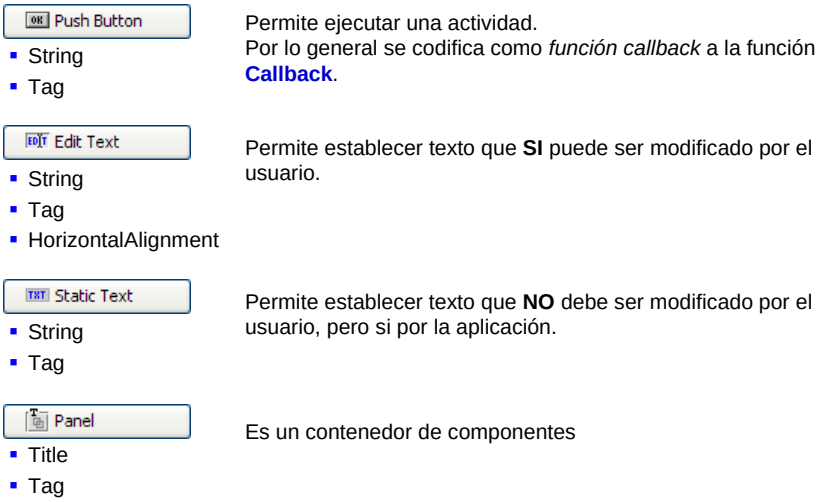


Para codificarlo procederemos de la siguiente manera:

1. En tiempo de diseño, damos clic derecho en el botón Aceptar cuyo Tag es psbAceptar.
2. Elegimos a la función Callback como respuesta.
3. Codificamos la función Callback del componente cuyo Tag es psbAceptar.
4. Cada vez que en tiempo de ejecución den clic en Aceptar, se ejecutará la función Callback asociada.
5. Pueden existir mas subfunciones de la función principal (no necesariamente callbacks)

7.11. Los Componentes Edit Text, Static Text, Panel y Push Button

A continuación se muestran los componentes básicos, una breve descripción y sus propiedades mas importantes:

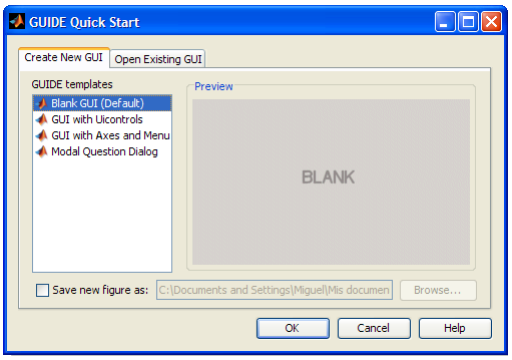


7.12. Resumen de pasos para la creación de una GUI con GUIDE

1. Iniciamos guide

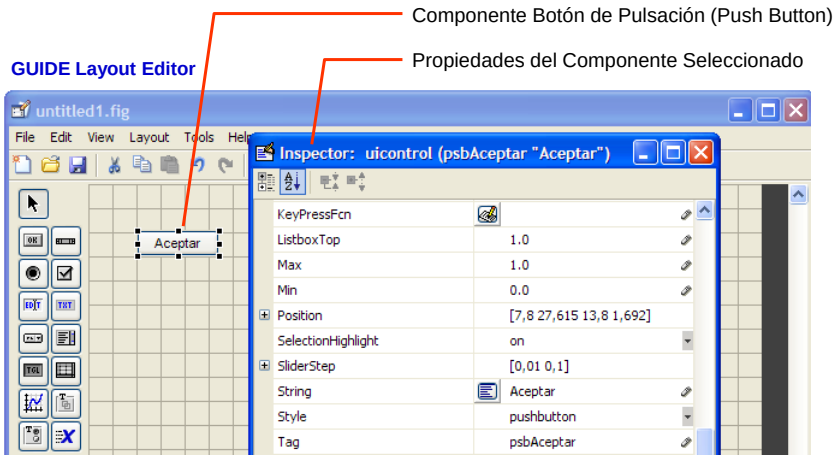
   >> guide

2. Seleccionamos una plantilla en la ventana GUIDE Quick Start

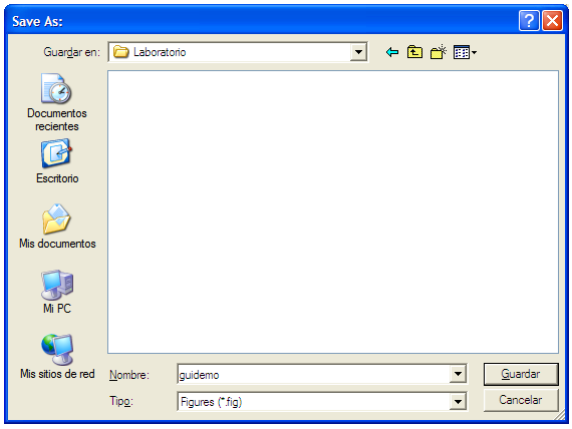


3. Se añaden los componentes necesarios arrastrándolos desde la paleta de componentes a la Ventana Figura (componente figure de la aplicación)

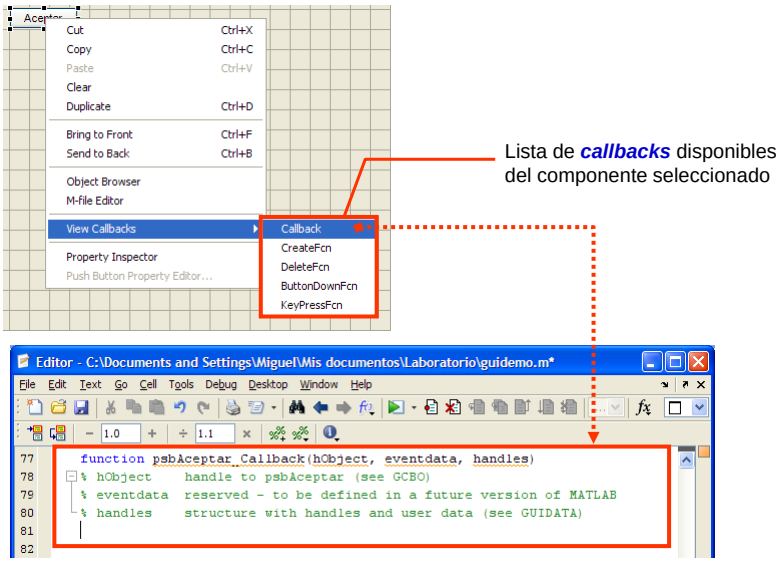
4. Se establecen las propiedades de cada componente



5. Se guarda la aplicación asignando un nombre con el cual GUIDE creará dos archivos: nombre.fig y nombre.m



6. Se codifica las devoluciones de llamada (callback) necesarias



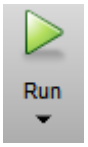
7. Se ejecuta la aplicación

   Existen varias formas:

   a) Invocando a la aplicación a través de su nombre desde la ventana de comandos

   b) Desde el Editor de archivos M

      1) Dando clic en



      2) Presionando [F5]

   c) Desde el GUIDE Layout Editor

      1) Dando clic en 
      2) Eligiendo la opción **Run** del menú **Tools**