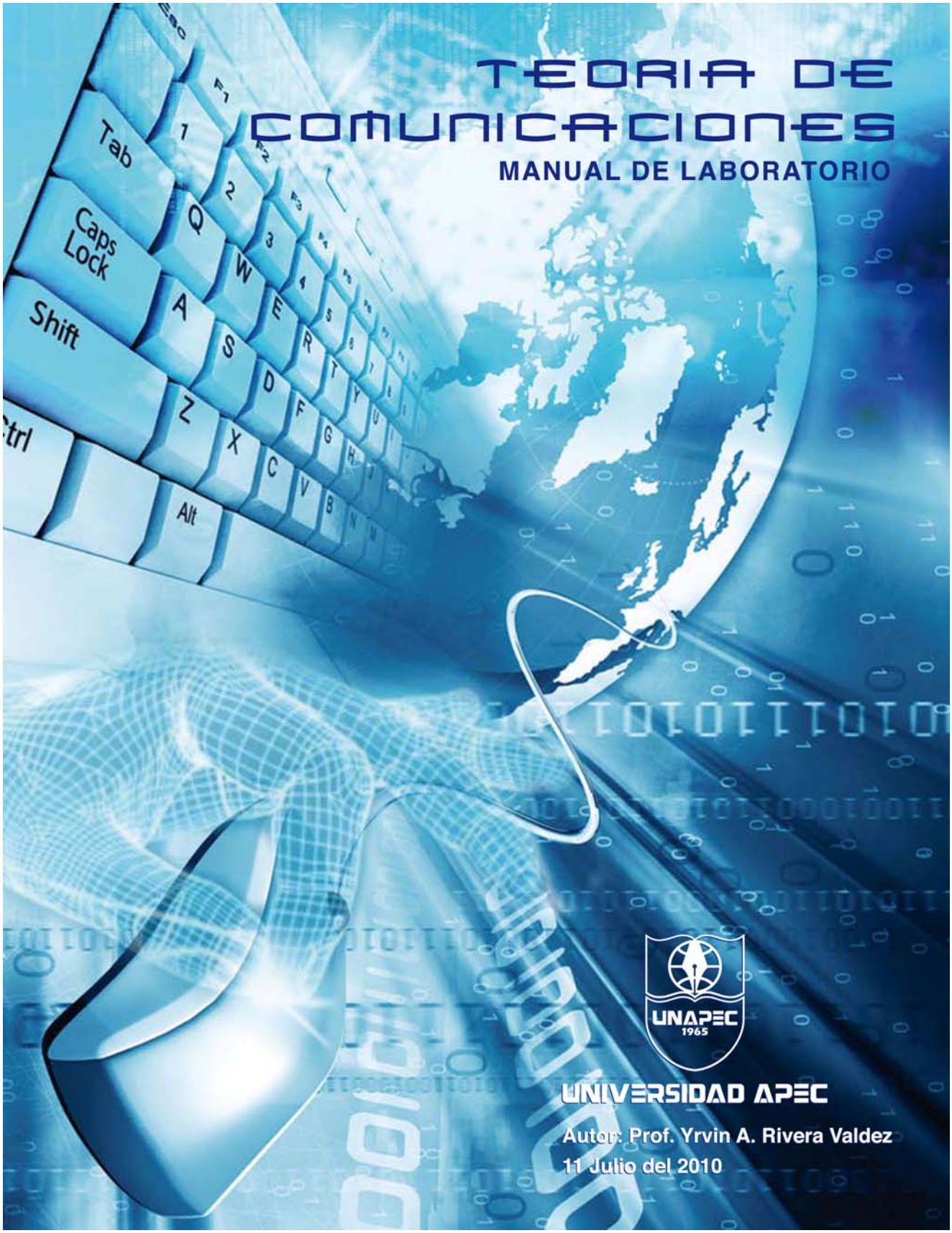


Universidad Acción Pro Educación y Cultura

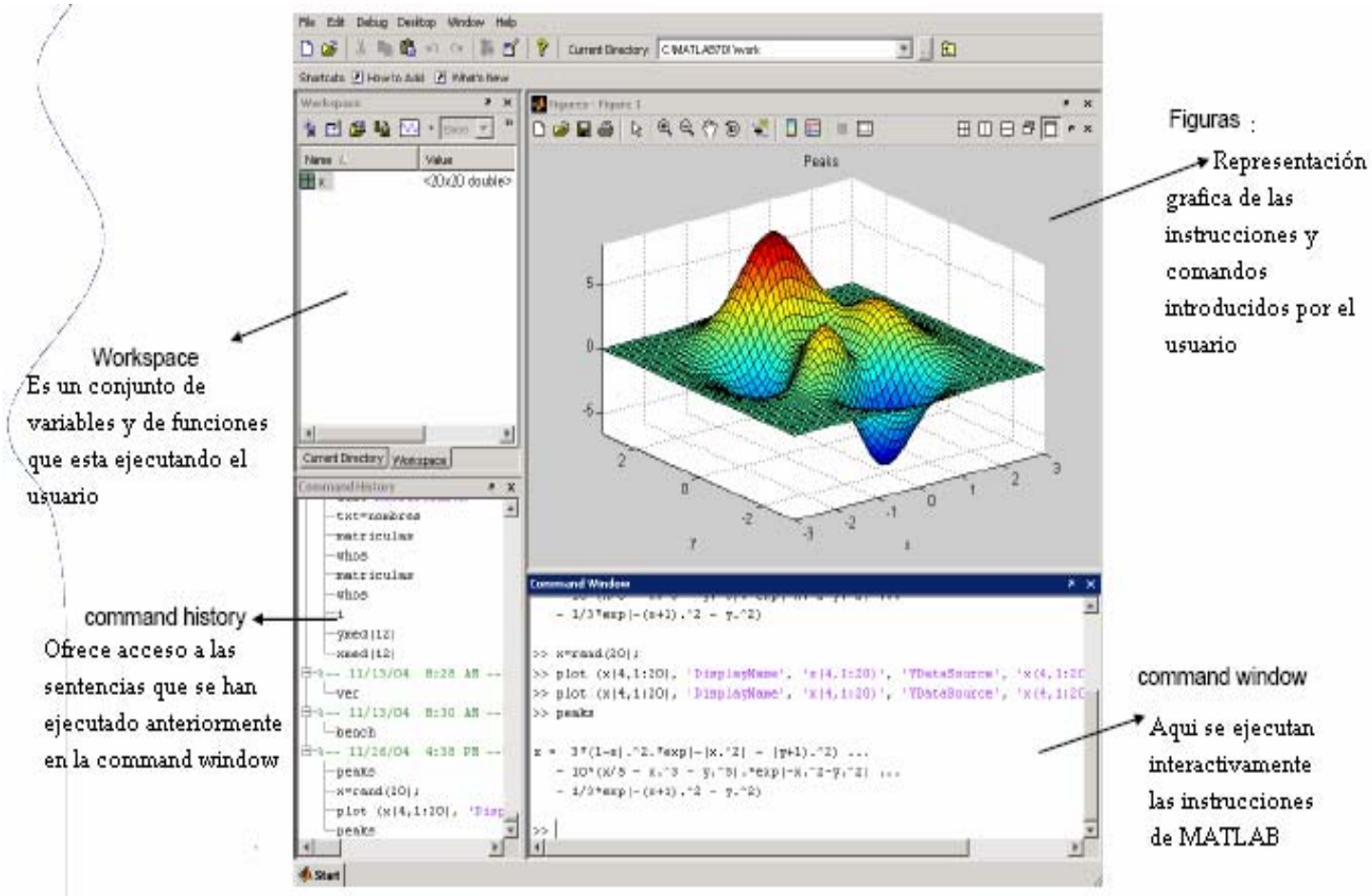
Introducción:

Este manual se ha elaborado con el propósito de proveer al estudiante las destrezas y conocimientos necesarios para el diseño y análisis de sistemas de señales y comunicaciones electrónicas, mediante herramientas de diseño electrónico asistido por computadoras. Esta asignatura de laboratorio de Teoría de Comunicaciones es la primera entrega de una serie de experimentos en el área de las comunicaciones electrónica.

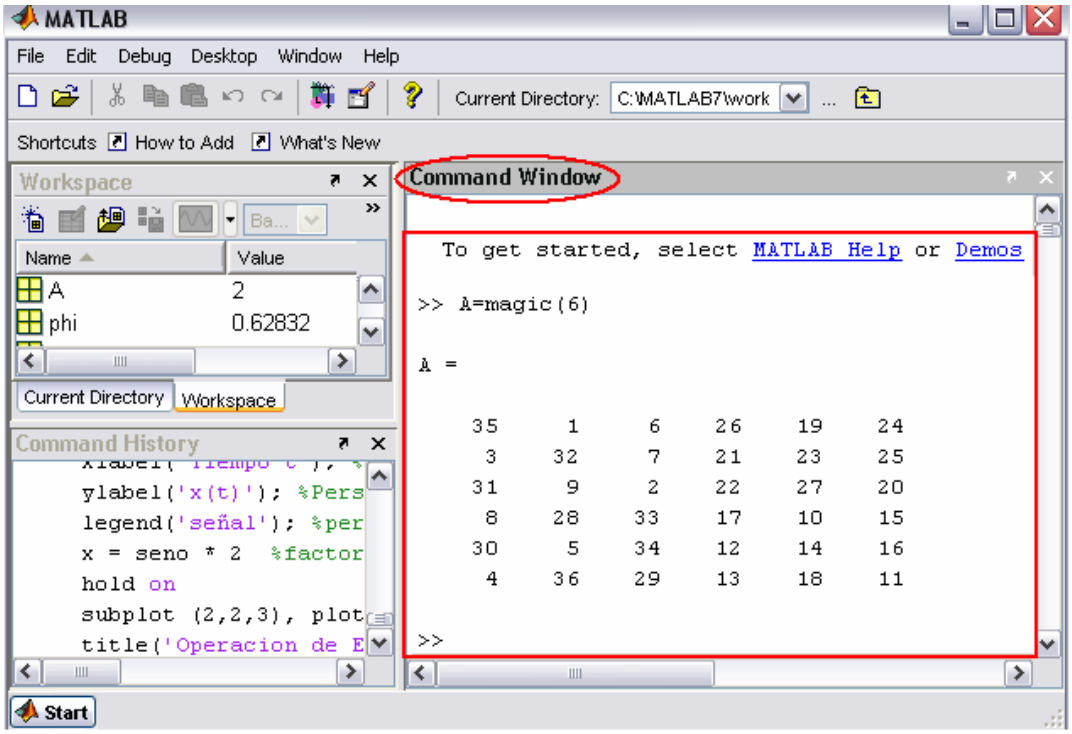
Al finalizar este programa de prácticas, el estudiante estará en condiciones de representar señales y sistemas de comunicaciones utilizando Matlab. Podrá realizar simulaciones de sistemas específicos de comunicaciones y establecer diferencias en el funcionamiento de los mismos. Este manual de laboratorio que sirve de guía, en el aparecen preguntas que deben ser contestadas por los estudiantes. El instructor facilita la realización de las prácticas, orientando a los estudiantes. Las prácticas están basadas en experimentos donde se utiliza el software Matlab.

Como premisa es necesario recordar algunos puntos básicos de Matlab

Entorno de trabajo del MATLAB V.7



Procederemos a describir cada una de las principales herramientas que posee el entorno de trabajo de MATLAB.



Siendo la parte mas importante de la ventana inicial el **Command Window**, la cual, aparece en la parte derecha. En esta sub-ventana es donde se ejecutan los comandos de MATLAB, a continuación del ***prompt*** (aviso) característico (>>), que indica que el programa está preparado para recibir instrucciones.

En la pantalla mostrada en la Figura, se ha ejecutado el comando **A=magic(6)**, mostrándose a continuación el resultado proporcionado por MATLAB.

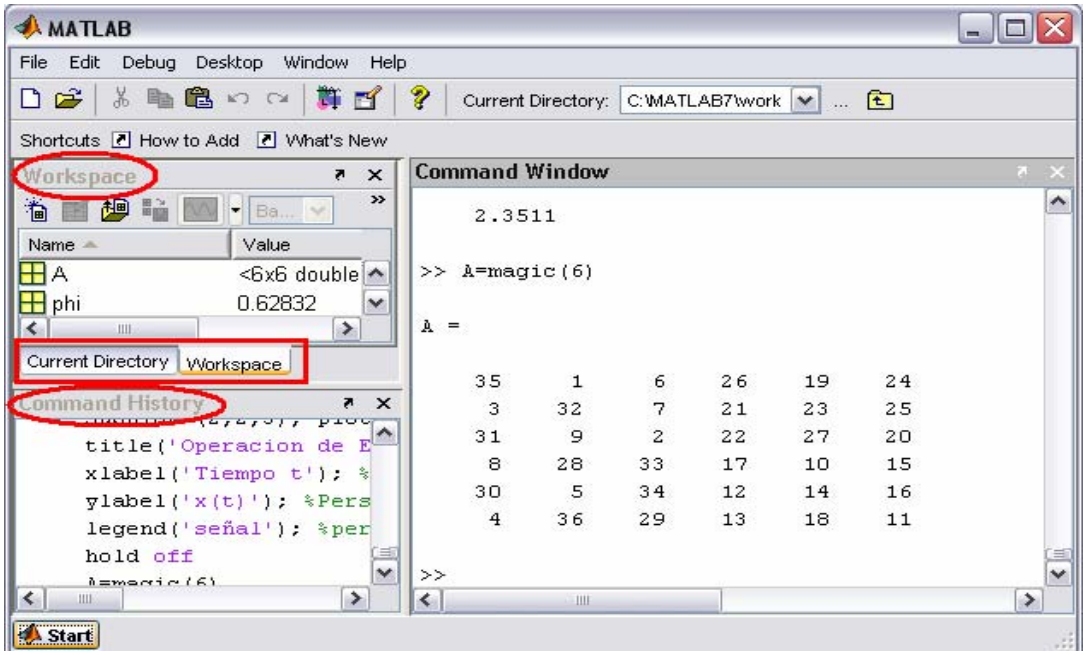


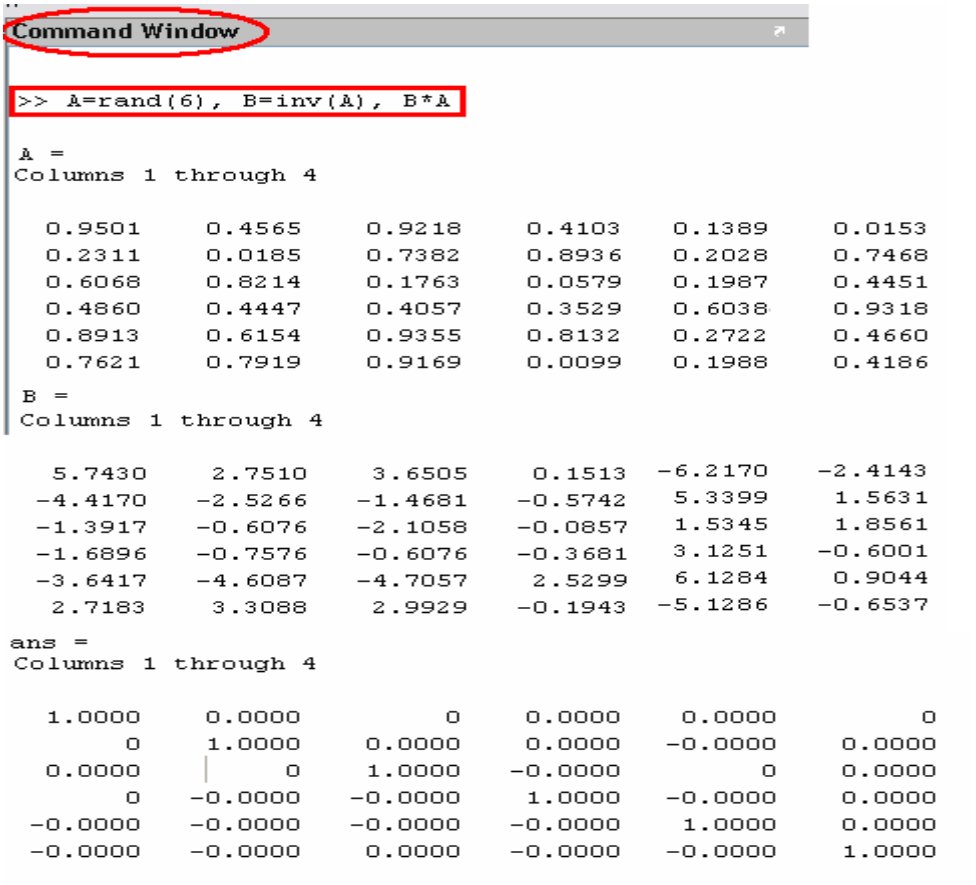
Figura 1. Ventana inicial de MATLAB 7.0.

En la parte superior izquierda de la pantalla aparecen dos ventanas también muy útiles: en la parte superior aparece la ventana ***Current Directory***, que se puede alternar con ***Workspace*** clicando en la pestaña correspondiente.

La ventana ***Current Directory*** muestra los ficheros del directorio activo o actual. El directorio activo se puede cambiar desde la ***Command Window***, o desde la propia ventana (o desde la barra de herramientas, debajo de la barra de menús) con los métodos de navegación de directorios propios de ***Windows***. Clicando dos veces sobre alguno de los ficheros ****.m*** del directorio activo se abre el ***editor de ficheros*** de MATLAB, herramienta fundamental para la programación sobre la que se volverá en las próximas páginas.

El ***Workspace*** contiene información sobre todas las variables que se hayan definido en esta sesión y permite ver y modificar las matrices con las que se esté trabajando.
En la parte inferior derecha aparece la ventana ***Command History*** que muestra los últimos comandos ejecutados en la ***Command Window***. Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos. Clicando sobre un comando con el botón derecho del ratón se muestra un menú contextual con las posibilidades disponibles en ese momento. Para editar uno de estos comandos hay que copiarlo antes a la ***Command Window***.

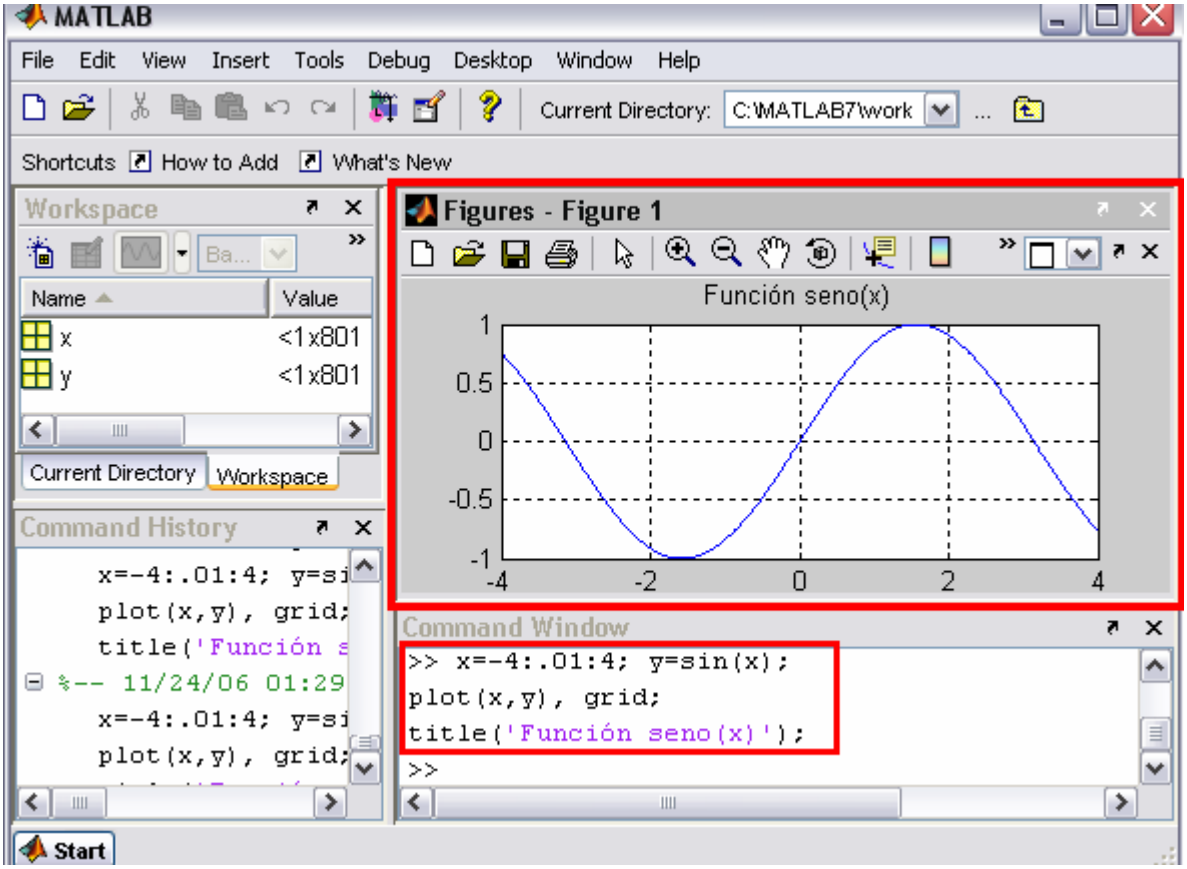
Para apreciar desde el principio la potencia e iniciar nuestra practica de MATLAB, se puede comenzar por escribir en la ***Command Window*** la siguiente línea, **`A=rand(6), B=inv(A), B*A`**
, a continuación del *prompt* es decir, (>>). Al final hay que pulsar ***intro***.



En realidad, en la línea de comandos anterior se han escrito tres instrucciones diferentes, separadas por comas. Como consecuencia, la respuesta del programa tiene tres partes también, cada una de ellas correspondiente a una de las instrucciones. Con la primera instrucción se define una matriz cuadrada (6×6) llamada **A**, cuyos elementos son números aleatorios entre cero y uno (aunque aparezcan
sólo 4 cifras, han sido calculados con 16 cifras de precisión).
En la segunda instrucción se define una matriz **B** que es igual a la inversa de **A**. Finalmente se ha multiplicado **B** por **A**, y se comprueba que el resultado es la matriz unidad.
Es con grandes matrices o grandes sistemas de ecuaciones como MATLAB obtiene toda la potencia del ordenador.

Otro de los puntos fuertes de MATLAB son **los gráficos**, que se verán con más detalle en las proximas practicas. A título de ejemplo, se puede teclear la siguiente línea y pulsar ***intro***

```
>> x=-4:.01:4; y=sin(x); plot(x,y), grid, title('Función seno(x)')
```



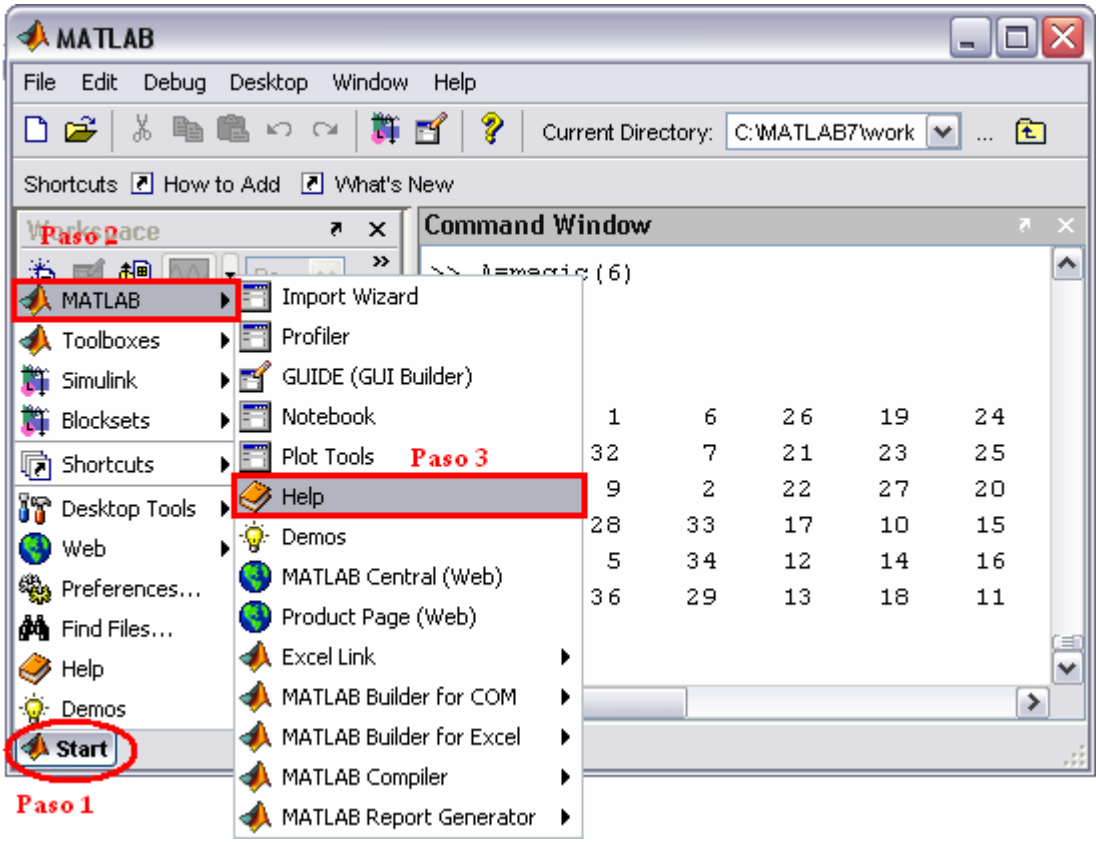
En la Figura se puede observar que se abre una nueva ventana en la que aparece representada la función ***sin(x)***. Esta figura tiene un título "Función seno(x)" y una cuadrícula o "grid".

En realidad la línea anterior contiene también varias instrucciones separadas por comas o puntos y comas. En la primera se crea un vector **x** con 801 valores reales entre -4 y 4, separados por una centésima.

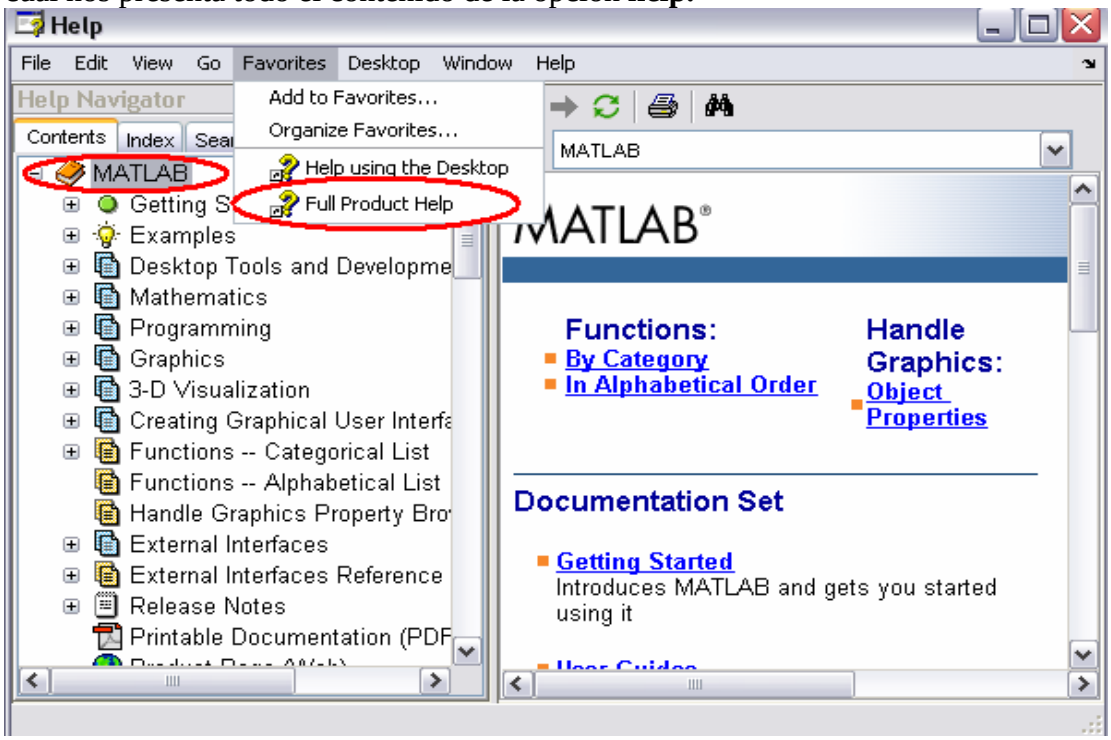
A continuación se crea un vector **y**, cada uno de cuyos elementos es el seno del correspondiente elemento
del vector **x**. Después se dibujan los valores de **y** en ordenadas frente a los de **x** en abscisas.
Las dos últimas instrucciones establecen la cuadrícula y el título.

Uso del *Help*

MATLAB 7.0 dispone de un excelente ***Help*** con el que se puede encontrar la información que se desee. La Figura muestra los pasos a seguir para ingresar al menú del help.



Cuando damos un click con el puntero del Mouse sobre la opcion **HELP.** Este nos abre otra ventana, la cual nos presenta todo el contenido de la opcion **help.**



1. ***Full Product Family Help***, Se abre la ventana de la Figura, en la que se puede buscar información general sobre MATLAB o sobre otros productos de la familia a los que se tenga acceso. La forma de la ventana de ayuda es típica y común con otros niveles de ayuda. La mayor parte de las páginas de ayuda están en formato HTML.

2. ***MATLAB Help.*** Es por medio de la cual, se puede buscar ayuda general sobre MATLAB o sobre la función o el concepto que se desee. La portada de esta ayuda tiene **tres** capítulos principales:

Functions, que contiene información de referencia sobre las funciones por orden alfabético o por categorías; ***Ande Graphics***, que permite acceder a información concreta sobre las distintas propiedades de los objetos gráficos; ***Documentation Set***, que da acceso a versiones completas de los manuales del programa en formato de pantalla fácilmente navegable (con apartados de *Getting Started*, *User Guides*, *Programming Tips* y *Examples in Documentation*).

Product Demos (con una colección de ejemplos programados que se pueden ejecutar y cuyo código se puede examinar para ver cómo están programados), ***What's New*** (con las novedades de esta versión respecto a la anterior), ***Printing the Documentation Set*** (que permite abrir documentos PDF (*Portable Document Format*), que se corrresponden con las versiones en papel de los manuales del programa, y que precisan del programa ***Adobe Acrobat Reader 5.0*** o superior.) y un apartado final sobre ***The MathWorks Web Site Resources*** (que permite acceder a una amplísima colección de informaciones adicionales disponibles en la web de la empresa que ha desarrollado MATLAB).

En la parte izquierda de la ventana, cuando está seleccionada la pestaña ***Contents***, aparece un índice temático estructurado en forma de árbol que puede ser desplegado y recorrido con gran facilidad. Las restantes pestañas de esta ventana dan acceso a un índice por palabras (***Index***), a un formulario de búsqueda (***Search***) y a la colección de ejemplos ya programados antes citada (***Demos***).

3. ***Using the Desktop***. Se abre una ventana de ayuda con un formato similar a las de las Figuras anteriores con información detallada sobre cómo utilizar y configurar el entorno de desarrollo o ***Desktop***. Las distintas herramientas disponibles se describen sucesivamente. Cada página dispone de flechas y enlaces que permiten ir a la página siguiente o volver a la anterior. Es posible también imprimir aquellas páginas que se desee consultar o archivar sobre papel. Una característica muy importante es la posibilidad de organizar las ventanas con gran flexibilidad, agrupándolas o independizándoles según los propios gustos o deseos.

4. ***Using the Command Window***. Esta opción del menú ***Help*** da acceso a la información necesaria para aprovechar las capacidades de la ***Command Window***, que es el corazón de MATLAB.

5. ***Web Resources***. La **¡Error! No se encuentra el origen de la referencia.** muestra algunas direcciones
de Internet con información interesante sobre MATLAB. Todas ellas corresponden a distintas secciones de la web de The Mathworks (la empresa que desarrolla y comercializa MATLAB), cuya página de inicio se muestra en primer lugar.

6. ***Check for Updates***. MATLAB se conecta con The Mathworks y comprueba si has versiones más recientes de los productos instalados. Si se es un usuario registrado, es posible descargar las versiones más actuales.

7. ***Demos***. Se abre una ventana como la mostrada en la Figura, que da acceso a un buen número de ejemplos resueltos con MATLAB, cuyos resultados se presentan gráficamente de diversas

formas. Es muy interesante recorrer estos ejemplos para hacerse idea de las posibilidades del programa, tanto en cálculo como en gráficos. Es asimismo muy instructivo analizar los ficheros ****.m*** de los ejemplos de características similares a las de la aplicación de se desea desarrollar. Además, de una forma muy inmediata, es posible también recurrir al ***Help*** desde la línea de comandos de la ***Command Window***. Se aconseja practicar un poco al respecto. Por ejemplo, obsérvese la respuesta a los siguientes usos del comando ***help***:

```
>> help
>> help lang
```

COMMAND WINDOW

Ésta es la ventana en la que se ejecutan interactivamente las instrucciones de MATLAB y en donde se muestran los resultados correspondientes, si es el caso. En cierta forma es ***la ventana más importante*** y la única que existía en las primeras versiones de la aplicación. En esta nueva versión se han añadido algunas mejoras significativas, como las siguientes:

1. Se permiten líneas de comandos muy largas que automáticamente siguen en la línea siguiente al llegar al margen derecho de la ventana. Para ello hay que activar la opción ***Wrap Lines***, en el menú ***File/Preferences/Command Window***.

2. Clicando con el botón derecho sobre el nombre de una función que aparezca en esta ventana se tiene acceso a la página del ***Help*** sobre dicha función. Si el código fuente (fichero ****.m***) está disponible, también se puede acceder al fichero correspondiente por medio del ***Editor/Debugger***.

3. Comenzando a teclear el nombre de una función y pulsando la tecla ***Tab***, MATLAB ***completa automáticamente*** el nombre de la función, o bien muestra en la línea siguiente todas las funciones disponibles que comienzan con las letras tecleadas por el usuario.

4. Cuando al ejecutar un fichero ****.m*** se produce un error y se obtiene el correspondiente mensaje en la ***Command Window***, MATLAB muestra mediante un subrayado un ***enlace a la línea del fichero fuente*** en la que se ha producido el error. Clicando en ese enlace se va a la línea correspondiente del fichero por medio del ***Editor/Debugger***.

COMMAND HISTORY BROWSER

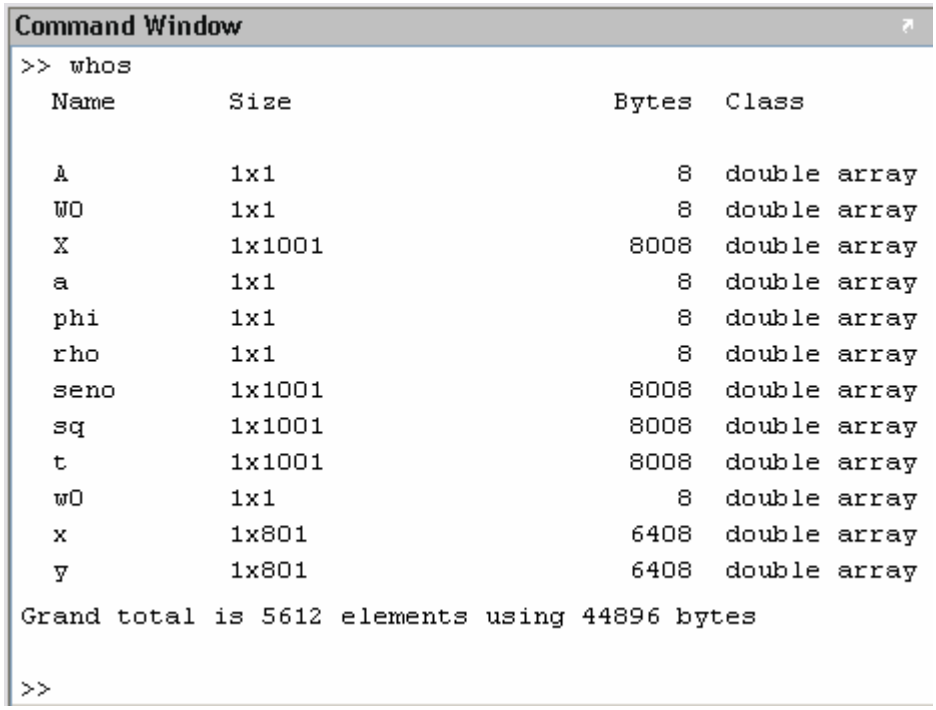
La ventana ***Command History*** ofrece acceso a las sentencias que se han ejecutado anteriormente en la ***Command Window***. Estas sentencias están también accesibles por medio de las teclas ↑ y ↓ como en las versiones anteriores, pero esta ventana facilita mucho el tener una visión más general de lo hecho anteriormente y seleccionar lo que realmente se desea repetir.

Las sentencias ejecutadas anteriormente se pueden volver a ejecutar mediante un doble clic o por medio del menú contextual que se abre al clicar sobre ellas con el botón derecho. También se pueden copiar y volcar sobre la línea de comandos, pero se ha de copiar toda la línea, sin que se admita la copia de un fragmento de la sentencia. Existen opciones para borrar algunas o todas las líneas de esta ventana. Se puede también hacer un ***profile*** (evaluar la eficiencia relativa) de una sentencia o de un grupo de sentencias.

WORKSPACE BROWSER Y ARRAY EDITOR

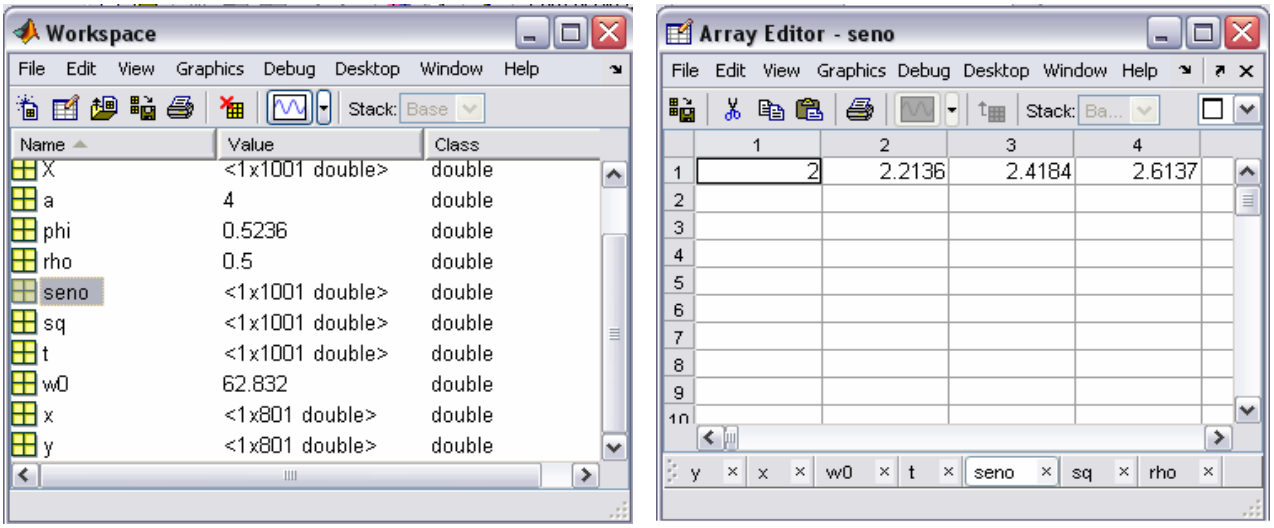
El espacio de trabajo de MATLAB (***Workspace***) es el conjunto de variables y de funciones de usuario

que en un determinado momento están definidas en la memoria del programa o de la función que se está ejecutando. Para obtener información sobre el ***Workspace*** desde la línea de comandos se pueden utilizar los comandos ***who*** y ***whos***. El segundo proporciona una información más detallada que el primero. Por ejemplo, una salida típica del comando ***whos*** es la siguiente:
>>whos



Éstas son las variables del ***espacio de trabajo base*** (el de la línea de comandos de MATLAB). Más adelante se verá que ***cada función tiene su propio espacio de trabajo***, con variables cuyos nombres no interfieren con las variables de los otros espacios de trabajo.

La ventana ***Workspace*** constituye un entorno gráfico para ver las variables definidas en el espacio de trabajo. Se activa con el comando ***View/Workspace***.



La Figura muestra el aspecto inicial de la ventana ***Workspace*** cuando se abre desde un determinado programa. Haciendo doble clic por ejemplo sobre la matriz **BARS** aparece una nueva ventana (o pestaña, si la ventana ya existía) del ***ArrayEditor***, en la que se muestran y pueden ser modificados los elementos de dicha matriz.

Es importante insistir en que cada una de las funciones de MATLAB tiene su propio espacio de trabajo, al que en principio sólo pertenecen las variables recibidas como argumentos o definidas dentro de la propia función.
En la barra de herramientas de la ventana ***Workspace*** aparece una lista desplegable llamada ***Stack***, con los espacios de trabajo del programa actual. Hay que tener en cuenta que cuando se termina de ejecutar una función y se devuelve el control al programa que la había llamado, las variables definidas en la función dejan de existir (salvo que se hayan declarado como ***persistent***) y también deja de existir su espacio de trabajo.

Si se desean examinar otras matrices y/o vectores, al hacer doble clic sobre ellas el ***Array Editor*** las muestra en la misma ventana como subventanas con una pestaña diferente. Clicando con el botón derecho sobre alguna de las variables del ***Workspace Browser*** se abre un menú contextual que ofrece algunas posibilidades interesantes, como por ejemplo la de ***representar gráficamente*** dicha variable.

El ***Array Editor*** no sólo permite ver los valores de los elementos de cualquier matriz o vector definido
en el programa: es también posible ***modificar estos valores*** clicando sobre la celda correspondiente. La ventana del ***Array Editor*** incluye una lista desplegable en la que se puede elegir el formato en el que se desea ver los datos.

El ***Array Editor*** es muy útil también para entender bien ciertos algoritmos, ejecutando paso a paso un programa y viendo cómo cambian los valores de las distintas variables. Es posible aparcar o situar las ventanas o pestañas del ***Array Editor*** en la misma ventana del ***Editor/Debugger***.

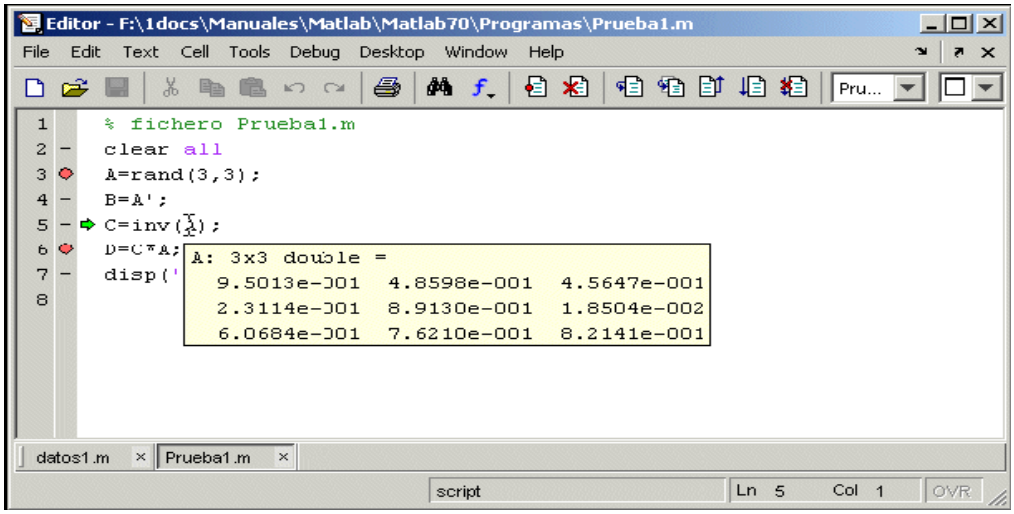
EL EDITOR/DEBUGGER

En MATLAB tienen particular importancia los ya citados ***ficheros-M*** (o ***M-files***). Son ficheros de texto ASCII, con la extensión ****.m***, que contienen ***conjuntos de comandos*** o ***definición de funciones*** (estos últimos son un poco más complicados y se verán más adelante). La importancia de estos ***ficheros-M*** es que al teclear su nombre en la línea de comandos y pulsar ***Intro***, se ejecutan uno tras otro todos los comandos contenidos en dicho fichero. El poder guardar instrucciones y grandes matrices
en un fichero permite ahorrar mucho trabajo de tecleado.

Aunque los ficheros ****.m*** se pueden crear con cualquier editor de ficheros ASCII tal como ***Notepad***, MATLAB dispone de un ***editor*** que permite tanto crear y modificar estos ficheros, como ejecutarlos paso a paso para ver si contienen errores (proceso de ***Debug*** o depuración).

La Figura muestra la ventana principal del ***Editor/Debugger***, en la que se ha tecleado un ***fichero-M*** llamado ***Prueba1.m***, que contiene un comentario y seis sentencias[5].
El ***Editor*** muestra con diferentes colores los diferentes tipos o elementos constitutivos de los comandos (en *verde* los comentarios, en *violeta* las cadenas de caracteres, etc.).
El ***Editor*** se preocupa también de que las comillas o paréntesis que se abren, no se queden sin el correspondiente elemento de cierre. Colocando el cursor antes o después de una apertura o cierre de corchete o paréntesis y pulsando las teclas (←) o (→), el ***Editor*** muestra con qué cierre o apertura de corchete o paréntesis se empareja el elemento considerado; si no se empareja con ninguno, aparece con una rayita de tachado.
Seleccionando varias líneas y clicando con el botón derecho aparece un menú contextual cuya sentencia

Comment permite entre otras cosas ***comentar con el carácter %*** todas las líneas seleccionadas. Estos comentarios pueden volver a su condición de código ejecutable seleccionándolos y ejecutando ***Uncomment*** en el menú contextual. Otra opción muy útil de ese menú contextual es ***Smart Indent***, que organiza el sangrado de los bucles y bifurcaciones de las sentencias seleccionadas.



La Figura corresponde a una ejecución de este fichero de comandos controlada con el ***Debugger***.

Dicha ejecución se comienza eligiendo el comando ***Run*** en el menú ***Debug***, pulsando la tecla **F5**, clicando en el botón ***Continue*** ( ) de la barra de herramientas del ***Editor*** o tecleando el nombre del fichero en la línea de comandos de la ***Command Window***. Los puntos rojos que aparecen en el margen izquierdo son ***breakpoints*** (puntos en los que se detiene la ejecución de programa); la ***flecha verde*** en el borde izquierdo indica la sentencia en que está detenida la ejecución (antes de ejecutar dicha sentencia); cuando el cursor se coloca sobre una variable (en este caso sobre **A**) aparece una pequeña ***ventana con los valores numéricos*** de esa variable, tal como se ve en la Figura.

En la Figura 21 puede apreciarse también que están activados los botones que corresponden al ***Debugger***.
El significado de estos botones, que aparece al colocar sobre ellos el cursor, es el siguiente:

Set/Clear Breakpoint. Coloca o borra un ***breakpoint*** en la línea en que está el cursor.

Clear All Breakpoints. Elimina todos los ***breakpoints*** que haya en el fichero.

Step. Avanzar un paso sin entrar en las funciones de usuario llamadas en esa línea.

Step In. Avanzar un paso, y si en ese paso hay una llamada a una función cuyo fichero ****.m*** está accesible, entra en dicha función.

Step Out. Salir de la función que se está ejecutando en ese momento.

Continue. Continuar la ejecución hasta el siguiente ***breakpoint***.

Quit Debugging. Terminar la ejecución del ***Debugger***.

Stack. En la parte derecha de la barra de herramientas aparece esta lista desplegable (visible en la Figura con las letras ***Pru...***) mediante la cual se puede elegir el *contexto*, es decir el ***espacio de trabajo*** o el ámbito de las variables que se quieren examinar.Ya se ha comentado que el espacio de trabajo base (el de las variables creadas desde la línea de comandos) y el espacio de trabajo de cada función son diferentes.

El ***Debugger*** es un programa que hay que conocer muy bien, pues es muy útil para detectar y corregir errores. Es también enormemente útil para aprender métodos numéricos y técnicos de programación. Para aprender a manejar el ***Debugger*** lo mejor es ***practicar***.

Introducción a las líneas de comandos, y a ejercicios prácticos donde pondremos en ejecución dichos comandos

Líneas de comentarios

Ya se ha indicado que para MATLAB el carácter ***tanto por ciento*** (%) indica comienzo de comentario. Cuando aparece en una línea de comandos, el programa supone que todo lo que va desde ese carácter hasta el fin de la línea es un comentario.
Más adelante se verá que los comentarios de los ficheros ****.m*** tienen algunas peculiaridades importantes,
pues pueden servir para definir ***help***'s personalizados de las funciones que el usuario vaya creando.

MATLAB permite ***comentar bloques de sentencias***, es decir, muchas sentencias contiguas de una vez. Una forma de hacerlo es seleccionar las sentencias que se desea comentar, clicar con el botón derecho, y elegir la opción ***Comment*** en el menú que se abre; las sentencias selecionadas se comentan individualmente con el carácter %. De forma similar se pueden eliminar los comentarios.
Otra forma de comentar bloques de sentencias (similar a la utilizada en C/C++ con /* y */) es encerrar las líneas que se desea inutilizar entre los caracteres **%{** y **%}**. Los bloques comentados pueden incluirse dentro de otros bloques comentados más amplios (bloques anidados).

OPERACIONES CON MATRICES Y VECTORES

Ya se ha comentado que MATLAB es fundamentalmente un programa para cálculo matricial. Inicialmente
se utilizará MATLAB como ***programa interactivo***, en el que se irán definiendo las matrices, los vectores y las expresiones que los combinan y obteniendo los resultados sobre la marcha. Si estos resultados son asignados a otras variables podrán ser utilizados posteriormente en otras expresiones. En este sentido MATLAB sería como una potente calculadora matricial (en realidad es esto y mucho más...). Antes de tratar de hacer cálculos complicados, la primera tarea será aprender a introducir matrices y vectores desde el teclado. Más adelante se verán otras formas más potentes de definir matrices y vectores.

Definición de matrices desde teclado

Como en casi todos los lenguajes de programación, en MATLAB las matrices y vectores son ***variables***
que tienen ***nombres***. Ya se verá luego con más detalle las reglas que deben cumplir estos nombres. Por el momento se sugiere que se utilicen letras ***mayúsculas para matrices*** y letras ***minúsculas para vectores y escalares*** (MATLAB no exige esto, pero puede resultar útil).
Para definir una matriz *no hace falta declararlas o establecer de antemano su tamaño* (de hecho, se puede definir un tamaño y cambiarlo posteriormente). MATLAB determina el número de filas y de columnas en función del número de elementos que se proporcionan (o se utilizan). ***Las matrices se definen o introducen por filas***6; los elementos de una misma fila están separados por ***blancos*** o ***comas***, mientras que las filas están separadas por pulsaciones ***intro*** o por caracteres ***punto y coma*** (;). Por ejemplo, el siguiente comando define una matriz **A** de dimensión (3×3):

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

La respuesta del programa es la siguiente:

```
A =

     1     2     3
     4     5     6
     7     8     9

>>
```

A partir de este momento la matriz **A** está disponible para hacer cualquier tipo de operación con ella (además de valores numéricos, en la definición de una matriz o vector se pueden utilizar expresiones y funciones matemáticas). Por ejemplo, una sencilla operación con **A** es hallar su ***matriz traspuesta***. En MATLAB el apóstrofo (') es el símbolo de ***transposición matricial***.
Para calcular **A'** (traspuesta de **A**) basta teclear lo siguiente (se añade a continuación la respuesta del programa):

```
>> A'

ans =

     1     4     7
     2     5     8
     3     6     9

>>
```

Como el resultado de la operación no ha sido asignado a ninguna otra matriz, MATLAB utiliza un nombre de variable por defecto (***ans***, de *answer*), que contiene el resultado de la última operación. La variable ***ans*** puede ser utilizada como operando en la siguiente expresión que se introduzca.
También podría haberse asignado el resultado a otra matriz llamada **B**:

```
>> B = A'

B =

     1     4     7
     2     5     8
     3     6     9

>>
```

Ahora ya están definidas las matrices **A** y **B**, y es posible seguir operando con ellas. Por ejemplo, se puede hacer el producto **B*A** (deberá resultar una matriz simétrica):

```
>> B * A

ans =

    66    78    90
    78    93   108
    90   108   126
```

En MATLAB se accede a los elementos de un vector poniendo el índice entre paréntesis (por ejemplo

x(3) ó **x(i)**). Los elementos de las matrices se acceden poniendo los dos índices entre paréntesis, separados por una coma (por ejemplo **A(1,2)** ó **A(i,j)**).

Las matrices ***se almacenan por columnas*** (aunque se introduzcan por filas, como se ha dicho antes), y teniendo en cuenta esto ***puede accederse a cualquier elemento de una matriz con un sólo subíndice***. Por ejemplo, si **A** es una matriz (3×3) se obtiene el mismo valor escribiendo **A(1,2)** que escribiendo **A(4)**.

Invertir una matriz es casi tan fácil como trasponerla. A continuación se va a definir una nueva matriz **A** -no singular- en la forma:

```
>>  A=[5 2 -2; 8 7 4; -4 5 1]

A =

     5     2    -2
     8     7     4
    -4     5     1
```

Ahora se va a calcular la inversa de **A** y el resultado se asignará a **B**. Para ello basta hacer uso de la función ***inv( )*** (la precisión o número de cifras con que se muestra el resultado se puede cambiar con el menú ***File/Preferences/General***):

```
>> B = inv(A)

B =

    0.0522    0.0482   -0.0884
    0.0964    0.0120    0.1446
   -0.2731    0.1325   -0.0763
```

Para comprobar que este resultado es correcto basta pre-multiplicar **A** por **B**;

```
>> B * A

ans =

    1.0000    0.0000    0.0000
    0.0000    1.0000    0.0000
    0.0000    0.0000    1.0000
```

De forma análoga a las matrices, es posible definir un ***vector fila*** **x** en la forma siguiente (si los tres números están separados por ***blancos*** o ***comas***, el resultado será un vector fila):

```
>> x=[10 20 30] % vector fila

x =

    10    20    30
```

Por el contrario, si los números están separados por ***intros*** o ***puntos y coma*** (;) se obtendrá un ***vector columna***:

```
>> y=[11; 12; 13] % vector columna

y =

    11
    12
    13
```

MATLAB tiene en cuenta la ***diferencia entre vectores fila y vectores columna***. Por ejemplo, si se intenta sumar los vectores **x** e **y** se obtendrá el siguiente mensaje de error:

```
>> x+y
??? Error using ==> +
Matrix dimensions must agree.
```

Operaciones con matrices

OPERADORES ARITMÉTICOS

MATLAB puede operar con matrices por medio de ***operadores*** y por medio de ***funciones***. Se han visto ya los operadores *suma* (+), *producto* (*) y *traspuesta* ('), así como la función *invertir* ***inv( )***. Los operadores matriciales de MATLAB son los siguientes:

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

Estos operadores se aplican también a las variables o valores escalares, aunque con algunas diferencias.

Todos estos operadores son coherentes con las correspondientes operaciones matriciales: no se puede por ejemplo sumar matrices que no sean del mismo tamaño. Si los operadores no se usan de modo correcto se obtiene un mensaje de error.

Los operadores anteriores se pueden aplicar también de modo ***mixto***, es decir con un operando escalar y otro matricial. En este caso la operación con el escalar se aplica a cada uno de los elementos de la matriz. Considérese el siguiente ejemplo:

```
>> A=[1 2; 3 4]
A =
     1     2
     3     4
>> A*2
ans =
     2     4
     6     8
>> A-4
ans =
    -3    -2
    -1     0
```

FUNCIONES DE LIBRERÍA

MATLAB tiene un gran número de funciones incorporadas. Algunas son ***funciones intrínsecas***, esto es, funciones incorporadas en el propio código ejecutable del programa. Estas funciones son particularmente rápidas y eficientes. Existen además funciones definidas en ficheros ****.m*** y ****.mex***[12] que vienen con el propio programa o que han sido aportadas por usuarios del mismo. Estas funciones extienden en gran manera las posibilidades del programa.

MATLAB dispone también de ficheros ****.p***, que son los ficheros ****.m*** pre-compilados con la función ***pcode***. Recuérdese que para que MATLAB encuentre una determinada función de usuario el correspondiente
fichero-M debe estar en el ***directorio actual*** o en uno de los directorios del ***search path***.

Características generales de las funciones de MATLAB

El concepto de función en MATLAB es semejante al de C y al de otros lenguajes de programación, aunque con algunas diferencias importantes. Al igual que en C, una función tiene ***nombre***, ***valor de retorno*** y ***argumentos***. Una función *se llama* utilizando su nombre en una expresión o utilizándolo como un comando más. Las funciones se pueden definir en ficheros de texto ****.m*** en la forma que se verá más adelante. Considérense los siguientes ejemplos de llamada a funciones:

```
>> [maximo, posmax] = max(x);
>> r = sqrt(x^2+y^2) + eps;
>> a = cos(alfa) - sin(alfa);
```

donde se han utilizado algunas funciones matemáticas bien conocidas como el cálculo del valor máximo, el seno, el coseno y la raíz cuadrada. Los ***nombres*** de las funciones se han puesto en negrita. Los ***argumentos*** de cada función van a continuación del nombre entre paréntesis (y separados por comas si hay más de uno). Los ***valores de retorno*** son el resultado de la función y sustituyen a ésta en la expresión donde la función aparece.

Una diferencia importante con otros lenguajes es que en MATLAB las funciones pueden tener ***valores de retorno matriciales múltiples*** (ya se verá que pueden recogerse en variables *ad hoc* todos o sólo parte de estos valores de retorno), como en el primero de los ejemplos anteriores. En este caso se calcula el elemento de máximo valor en un vector, y se devuelven dos valores: el valor máximo y la posición que ocupa en el vector. Obsérvese que los 2 valores de retorno ***se recogen entre corchetes***, separados por comas.

Una característica de MATLAB es que las funciones que no tienen argumentos no llevan paréntesis, por lo que a simple vista no siempre son fáciles de distinguir de las simples variables. En la segunda línea de los ejemplos anteriores, ***eps*** es una función sin argumentos, que devuelve la diferencia entre 1.0 y el número de coma flotante inmediatamente superior. En lo sucesivo el nombre de la función irá seguido de paréntesis si interesa resaltar que la función espera que se le pase uno o más argumentos.

Los nombres de las funciones de MATLAB ***no son palabras reservadas*** del lenguaje. Es posible crear una variable llamada ***sin*** o ***cos***, que ocultan las funciones correspondientes. Para poder acceder a las funciones hay que eliminar (***clear***) las variables del mismo nombre que las ocultan, o bien haber definido previamente una ***referencia a función*** (*function handle*).

MATLAB permite que una función tenga un ***número variable de argumentos y valores de retorno***, determinado sólo en tiempo de ejecución. Más adelante se verá cómo se hace esto.

MATLAB tiene diversos tipos de funciones. A continuación se enumeran los tipos de funciones más importantes, clasificadas según su finalidad:

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

A continuación se enumeran algunas características generales de todas las funciones de MATLAB:

– Los ***argumentos actuales***[13] de estas funciones pueden ser expresiones y también llamadas a otra función.

– Las funciones de MATLAB nunca devuelven modificadas las variables que se pasan como argumentos, a no ser que se incluyan también como valores de retorno. Si el usuario las modifica dentro de la función, previamente se sacan copias de esas variables (se modifican las copias, no las variables originales). Se podría decir que los argumentos de las funciones de MATLAB siempre se pasan ***por valor***, ***nunca por referencia***.

– MATLAB admite valores de retorno matriciales múltiples. Por ejemplo, en el comando:

```
>> [V, D] = eig(A)
```

la función ***eig()*** calcula los valores y vectores propios de la matriz cuadrada **A**. Los vectores propios se devuelven como columnas de la matriz **V**, mientras que los valores propios son los elementos de la matriz diagonal **D**. En los ejemplos siguientes:

```
>> [xmax, imax] = max(x)
>> xmax = max(x)
```

puede verse que la misma función ***max()*** puede ser llamada recogiendo dos valores de retorno (el máximo elemento de un vector y la posición que ocupa) o un sólo valor de retorno (el máximo elemento).

– Las operaciones de suma y/o resta de una matriz con un escalar consisten en sumar y/o restar el escalar a todos los elementos de la matriz.

– Recuérdese que tecleando ***help nombre_funcion*** se obtiene de inmediato información sobre la función de ese nombre. En el ***Help Desk*** aparecen enlaces a "*Functions - By Cathegory*" y "*Functions – Alphabetical List*", en donde aparecen relaciones completas de las funciones disponibles en MATLAB.

Equivalencia entre comandos y funciones

Existe una equivalencia entre las funciones y los comandos con argumentos de MATLAB. Así, un comando en la forma,

```
>> comando arg1 arg2
```

es equivalente a una función con el mismo nombre que el comando a la que los argumentos se le pasan como cadenas de caracteres,

```
>> comando('arg1', 'arg2')
```

Esta dualidad entre comandos y funciones es sobre todo útil en programación, porque permite "*construir*" los argumentos con las operaciones propias de las cadenas de caracteres.

Funciones matemáticas elementales que operan de modo escalar

Estas funciones, que comprenden las funciones matemáticas trascendentales y otras funciones básicas, cuando se aplican a una matriz actúan sobre cada elemento de la matriz como si se tratase de un escalar. Por tanto, se aplican de la misma forma a escalares, vectores y matrices. Algunas de las funciones de este grupo son las siguientes:

sin(x) seno
cos(x) coseno
tan(x) tangente
asin(x) arco seno
acos(x) arco coseno
atan(x) arco tangente (devuelve un ángulo entre $-\pi/2$ y $+\pi/2$)
atan2(x) arco tangente (devuelve un ángulo entre $-\pi$ y $+\pi$); se le pasan 2 argumentos, proporcionales al seno y al coseno
sinh(x) seno hiperbólico
cosh(x) coseno hiperbólico
tanh(x) tangente hiperbólica
asinh(x) arco seno hiperbólico
acosh(x) arco coseno hiperbólico
atanh(x) arco tangente hiperbólica
log(x) logaritmo natural
log10(x) logaritmo decimal
exp(x) función exponencial
sqrt(x) raíz cuadrada

sign(x) devuelve -1 si <0, 0 si =0 y 1 si >0. Aplicada a un número complejo, devuelve un vector unitario en la misma dirección
rem(x,y) resto de la división (2 argumentos que no tienen que ser enteros)
mod(x,y) similar a ***rem*** (Ver diferencias con el ***Help***)
round(x) redondeo hacia el entero más próximo
fix(x) redondea hacia el entero más próximo a 0
floor(x) valor entero más próximo hacia -∞
ceil(x) valor entero más próximo hacia +∞
gcd(x) máximo común divisor
lcm(x) mínimo común múltiplo
real(x) partes reales
imag(x) partes imaginarias
abs(x) valores absolutos
angle(x) ángulos de fase

Exploración de conceptos con MATLAB

En esta primera práctica utilizaremos MATLAB para explorar la generación de las señales elementales.

Como son:

- Señales Periódicas
- Señales Exponenciales
- Señales Senoidales
- Señales Senoidales Amortiguadas Exponencialmente.

MATLAB ofrece un conjunto de herramientas para procesamiento de señales, la cual tiene una gran variedad de funciones para generar señales, la mayor parte de las cuales requieren que empecemos con la presentación vectorial del tiempo **t (**Tiempo Continuo**)** o **n (**Tiempo Discreto**).** Para generar un vector **t** de valores de tiempo con **intervalo de muestreo** F de **1** ms sobre el intervalo de 0 a 1s, por ejemplo se utiliza el comando.

>> t = 0:.001:1; (Para Señales En Tiempo Continuo)

Esto corresponde a 1000 muestras de tiempo por cada segundo o una **velocidad de muestreo** de 1000 Hz. Para generar un vector **n** de valores de tiempo para señales en **tiempo discreto**, digamos, de **n** = 0 a **n** = 1000, utilizamos el comando.

>> n = 0 : 1000; (Para Señales En Tiempo Discreto)

En MATLAB, una señal en tiempo discreto se representa exactamente, debido a que los vectores de la señal se describen como los elementos de un vector. Por otra parte, MATLAB brinda solo una aproximación a señales en tiempo continuo. La aproximación consiste en un vector cuyos elementos individuales son muestras de la señal en tiempo continuo subyacente. Cuando se usa este enfoque aproximado, es importante que se elija el intervalo de muestreo F suficientemente pequeño para asegurar que las muestras capturen todos los detalles de la señal.

Práctica No. 1

- **Generación de Señales Periódicas**

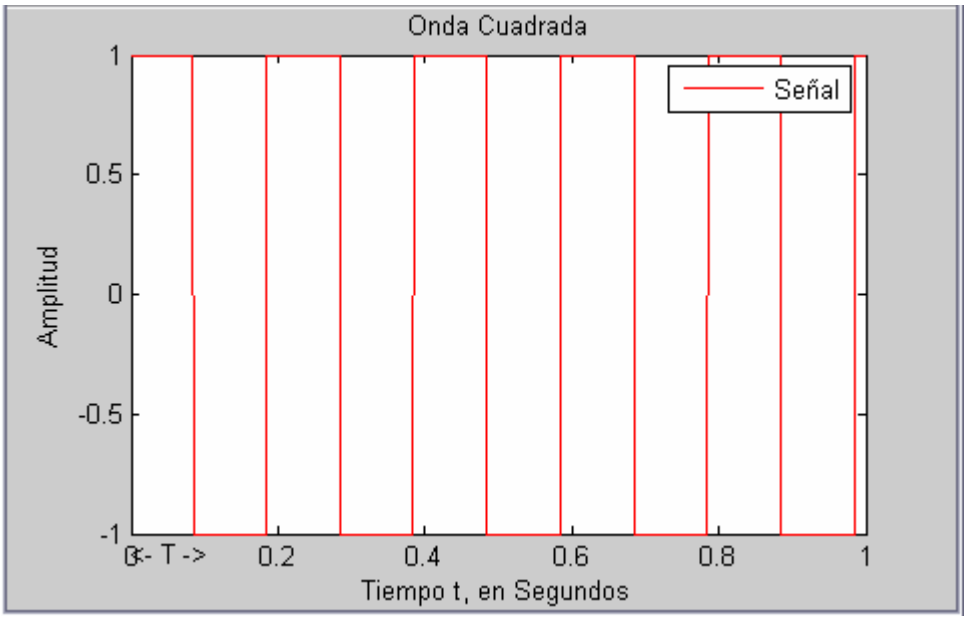
Generar señales periódicas tales como Ondas Cuadradas o Triangulares empleando MATLAB. Considere primero la generación de una onda cuadrada de amplitud **A**, frecuencia fundamental **w0** (medida en radianes por segundo), y ciclo de trabajo **rho**. Es decir, **rho** es la fracción de cada periodo en el cual la señal es positiva, **pi** es una función integrada en MATLAB que regresa el numero de punto flotante mas cercano a pi, el comando **plot** dibuja líneas que conectan los valores sucesivo de la señal y de ese modo brindan la apariencia de una señal en **tiempo continuo**.

Para general tal señal de Onda Cuadrada usamos el comando básico:

```
>> A*square(w0*t + rho);
```

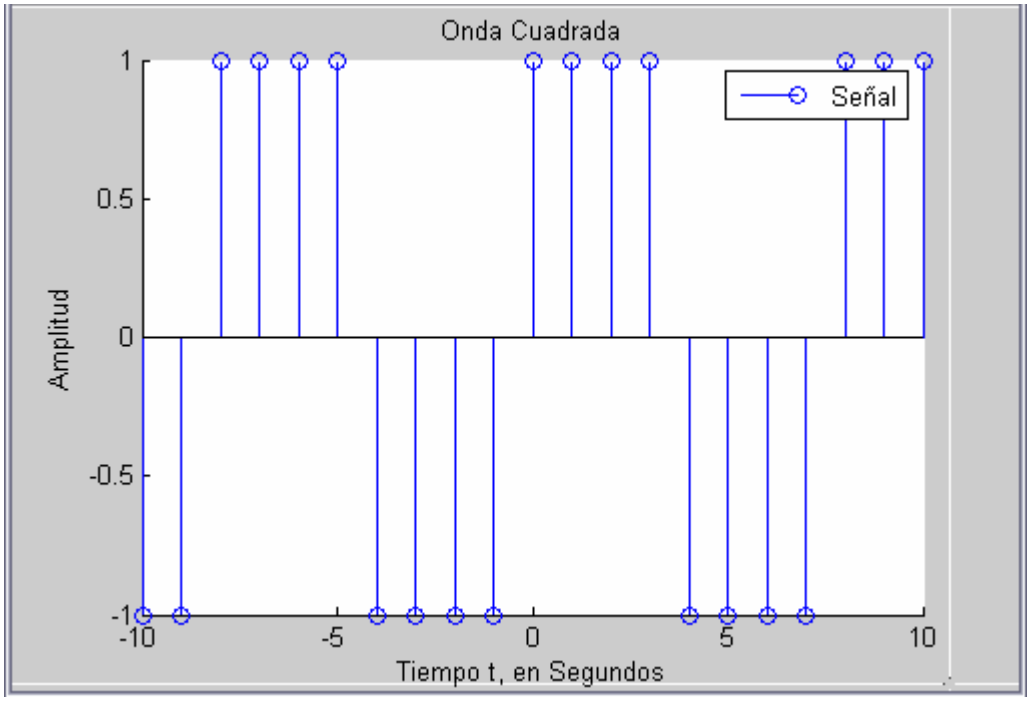
Para generar una Onda Cuadrada en Tiempo Continuo, empleamos el siguiente conjunto completo de comandos:

```
>>A = 1;  %Representa la Amplitud que en este caso es de 1
>>w0 = 10*pi; %Frecuencia Fundamental
>>rho = 0.5; % La fracción de cada periodo en el cual la señal es positive.
>>t = 0:.001:1;  % Intervalo de muestreo F   de 1 ms sobre el intervalo de 0 a 1s
>>sq = A*square(w0*t + rho);   % Para generar la Onda Cuadrada
>>subplot (2,2,1), plot(t,sq,'r');  % Para Graficar el t (intervalo de muestreo)  y sq (onda cuadrada)
>>title('Onda Cuadrada'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>gtext('<- T ->'); %Para agregar comentario
```



Para generar una Onda Cuadrada en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos:

```
>>A = 1;  %Representa la Amplitud que en este caso es de 1
>>omega = pi/4; %Frecuencia Fundamental
>>rho = 0.5; % Fracción de cada periodo en el cual la señal es positiva.
>>n = -10:10;  % Intervalo de muestreo F   de 1 ms sobre el intervalo de -10 a 10s
>>x = A*square(omega*n + rho);   % Para generar la Onda cuadrada en tiempo Discreto
>>subplot (2,2,4), stem(n,x,'r');  % Para Graficar el n (intervalo de muestreo)  y x (onda cuadrada)
>>title('Onda Cuadrada'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>gtext('<- T ->'); %Para agregar comentario
```



Para la generación de una **Onda Triangular** de amplitud **A,** frecuencia fundamental **w0**, y ancho **W**. Sea **T** el periodo de la onda triangular, con el primer valor máximo ocurriendo en **t = W.T**.

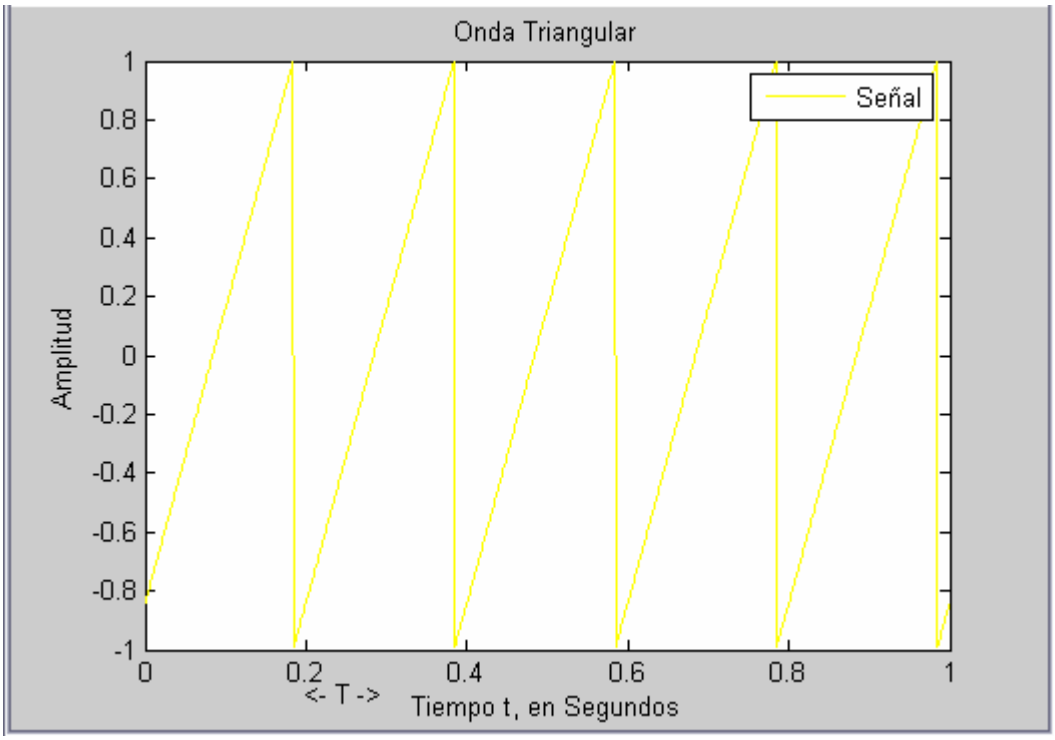
El comando básico para generar esta señal periódica es:

```
>> A*sawtooth(w0*t + W);
```

Para generar una Onda Triangular, empleamos el siguiente conjunto completo de comandos:

```
>>A = 1;  %Representa la Amplitud que en este caso es de 1
>>w0 = 10*pi; %Frecuencia Fundamental
>>W = 0.5; % Representa el Ancho
>>t = 0:.001:1;  % Intervalo de muestreo F   de 1 ms sobre el intervalo de 0 a 1s
>>tri = A*sawtooth(w0*t + W);   % Para generar la Onda Triangular
>>subplot (2,2,3), plot(t,tri,'y');  % Para Graficar el t y tri
>>title('Onda Triangular'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
```

```
>>legend('Señal'); %personifica la señal
>>gtext('<- T ->'); %Para agregar comentario
```

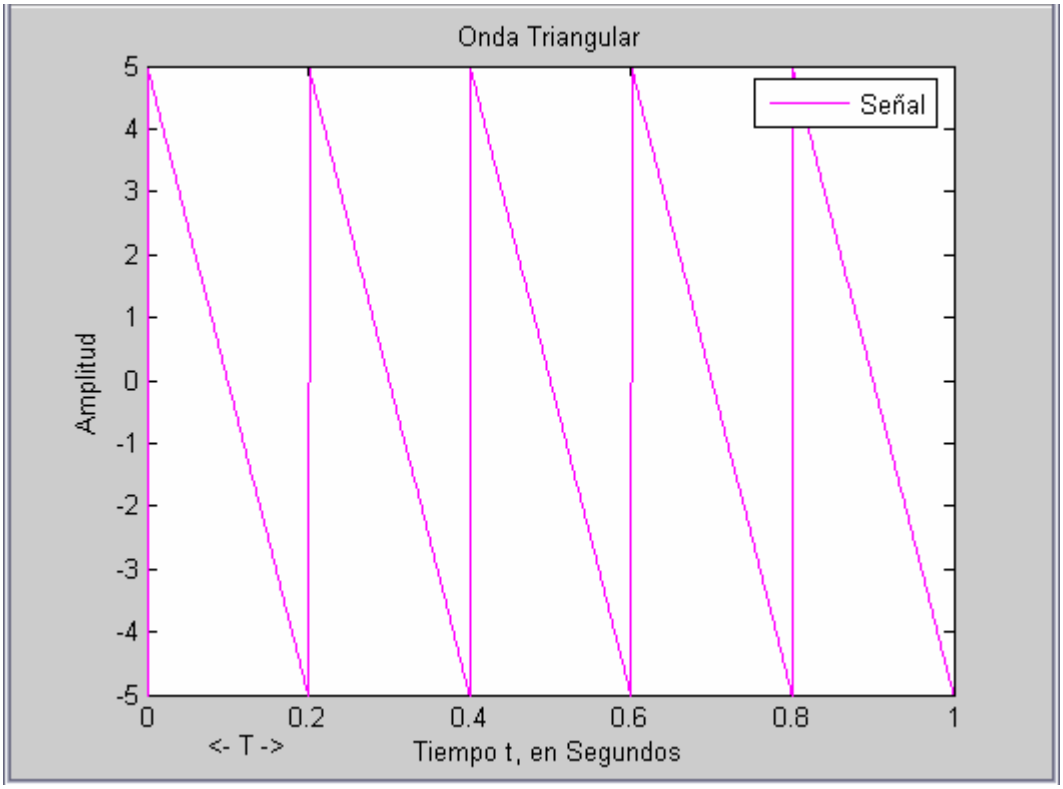


Ejercicio Práctico:

Genere una señal de Onda Triangular en Tiempo Continuo con corrimiento hacia la izquierda ósea negativo, la cual tendrá una A (amplitud) de 5, una Frecuencia Fundamental = 10*pi, y un ancho de 0.

Resp:
Para generar una Onda Triangular en Tiempo Continuo con corrimiento hacia la izquierda, utilizamos el siguiente conjunto de comandos. Tomando en cuenta que quien hace el corrimiento es La frecuencia fundamental. La cual si es positiva tendrá corrimiento hacia la derecha ósea positivo y si es negativa tendrá un corrimiento hacia la izquierda ósea negativo.

```
A = 5;  %Representa la Amplitud que en este caso es de 1
w0 = -10*pi; %Frecuencia Fundamental
W = 1; % Representa el Ancho
t = 0:.001:1;  % Intervalo de muestreo F   de 1 ms sobre el intervalo de 0 a 1s
tri = A*sawtooth(w0*t + W);   % Para generar la Onda Triangular
plot(t,tri,'m');  % Para Graficar el t y tri
title('Onda Triangular'); %Titulo de la hoja grafica
xlabel('Tiempo t, en Segundos'); %personifica el eje X
ylabel('Amplitud'); %Personifica el eje Y
legend('Señal'); %personifica la señal
gtext('<- T ->'); %Para agregar comentario
```

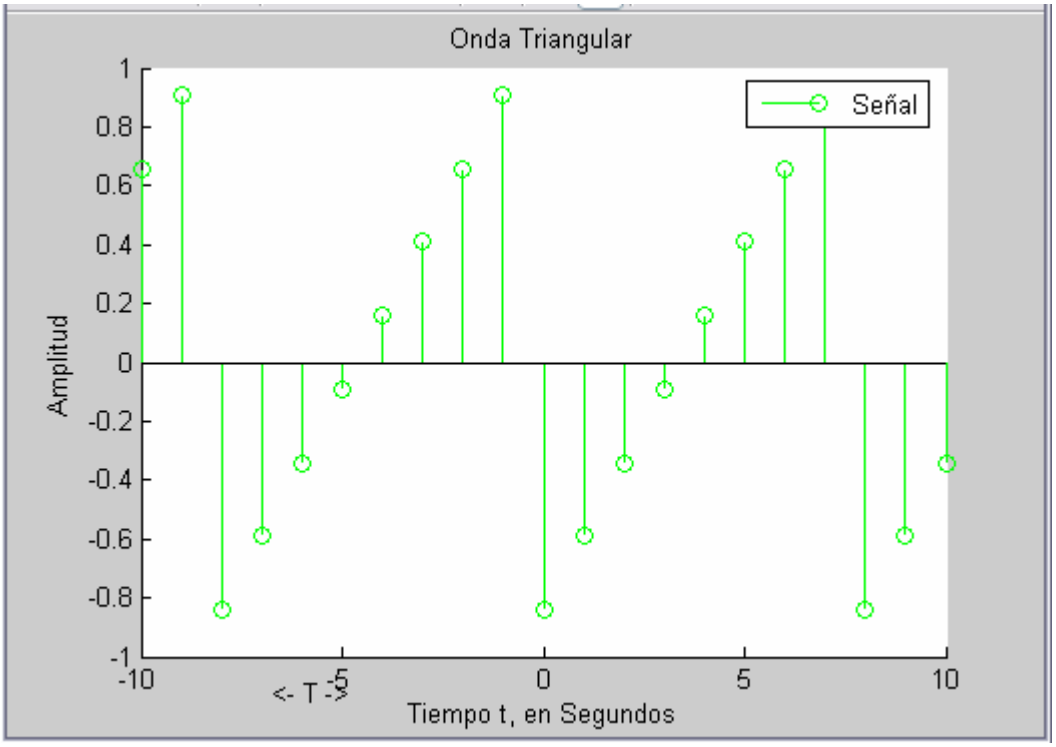


Ejercicio Práctico:

Genere una señal de Onda Triangular en Tiempo Discreto, la cual tendrá una A (amplitud) de 1, con una secuencia definida para n = -10:10, a una Frecuencia Fundamental = pi/4

Resp:
Para generar una Onda Triangular en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos:

```
>>A = 1;  %Representa la Amplitud que en este caso es de 1
>>omega = pi/4; %Frecuencia Fundamental
>>W = 0.5; % Fracción de cada periodo en el cual la señal es positiva.
>>n = -10:10;  % Intervalo de muestreo F   de 1 ms sobre el intervalo de -10 a 10s
>>tri = A*sawtooth(omega*n + W);   % Para generar la Onda triangular en tiempo Discreto
>>stem(n,tri,'g');  % Para Graficar el n   y tri
>>title('Onda Triangular'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>gtext('<- T ->'); %Para agregar comentario
```


Onda Triangular
Señal
Amplitud
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-10
<- T ->
-5
0
5
10
Tiempo t, en Segundos

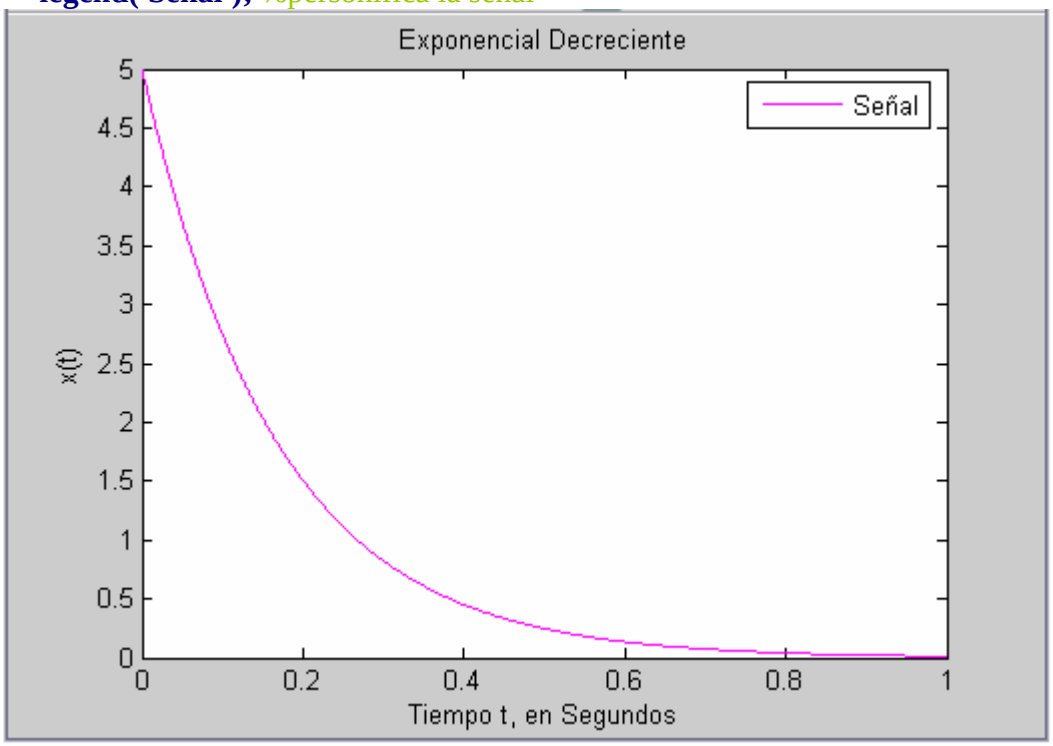
Práctica No. 2

- **Generación de Señales Exponenciales**

Al generar las señales exponenciales, tenemos **exponenciales que decaen** y **exponenciales crecientes.**
El comando MATLAB para generar una **Exponencial que decae es:**
>>B*exp (- a * t);
El comando MATLAB para generar una **Exponencial Creciente es:**
>>B*exp (a * t);
En ambos casos, el parámetro exponencial **a** es positivo.

Para generar una Señal Exponencial Decreciente en Tiempo Continuo, empleamos el siguiente conjunto completo de comandos:

```
>>B = 5;
>>a = 6;
>>t = 0:.001:1;
>>x = B*exp(-a*t);
>>subplot (2,2,1) , plot(t,x,'m')  % Para Graficar el t y tri
>>title('Exponencial Decreciente); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
```

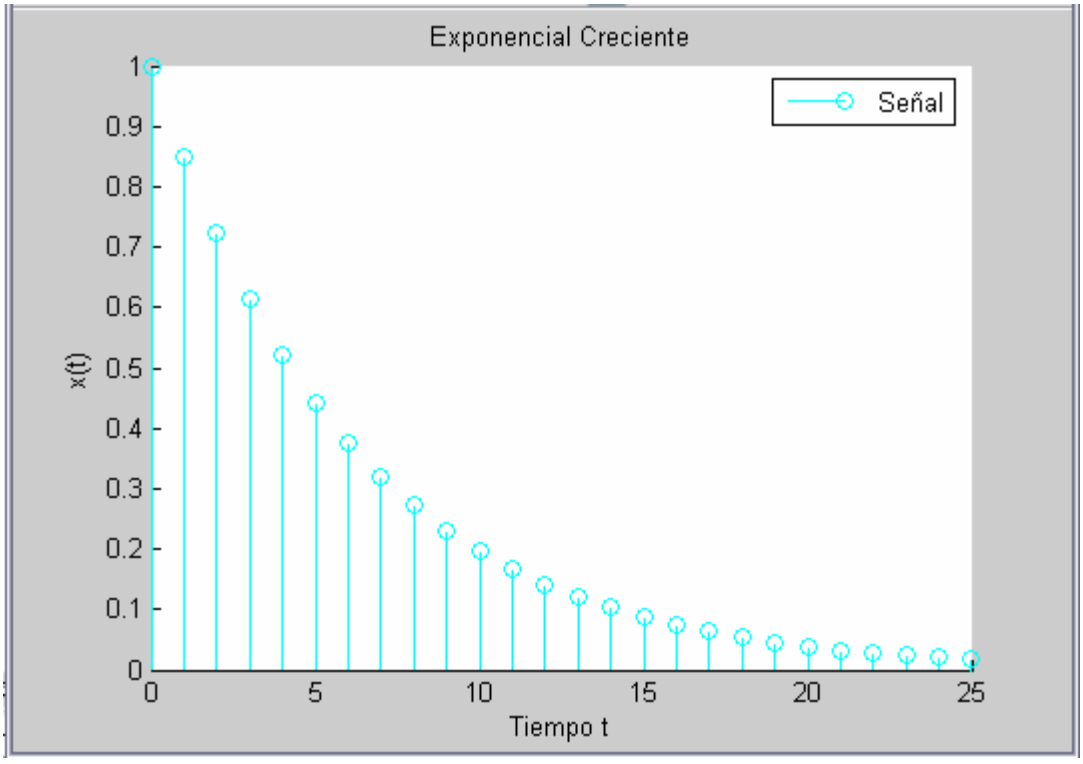


Ejercicio Práctico:

Genere una Señal Exponencial Decreciente en Tiempo Discreto, la cual tendrá una A (amplitud) de 1, Un intervalo de tiempo de n= -0:25 , y un ancho de 0.85

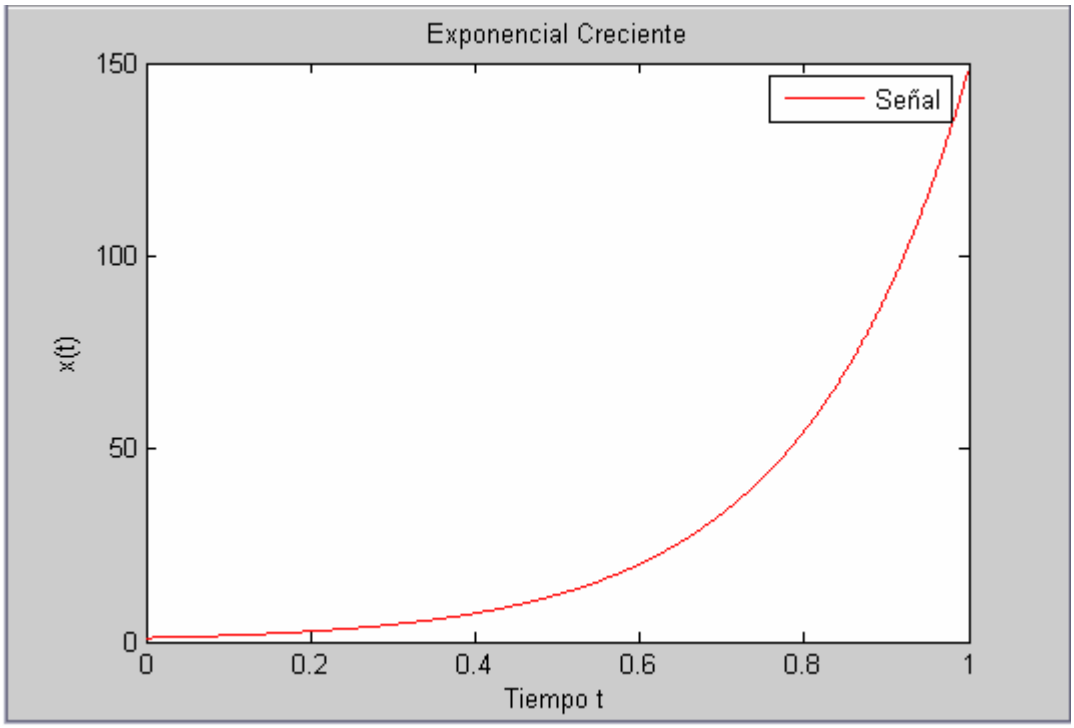
Para generar una Señal Exponencial Decreciente en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos, tomando en cuenta que para la representación de la señal exponencial en Tiempo Discreto utilizamos la siguiente formula **x = A(**altura**)*r(**ancho**).^n(**tiempo Discreto**)**:

```
>>A = 1;
>>r = 0.85;
>>n = -0:25;
>>x = A*r.^n;
>>stem(n,x,'c')  % Para Graficar el t y tri
>>title('Exponencial Creciente); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
```



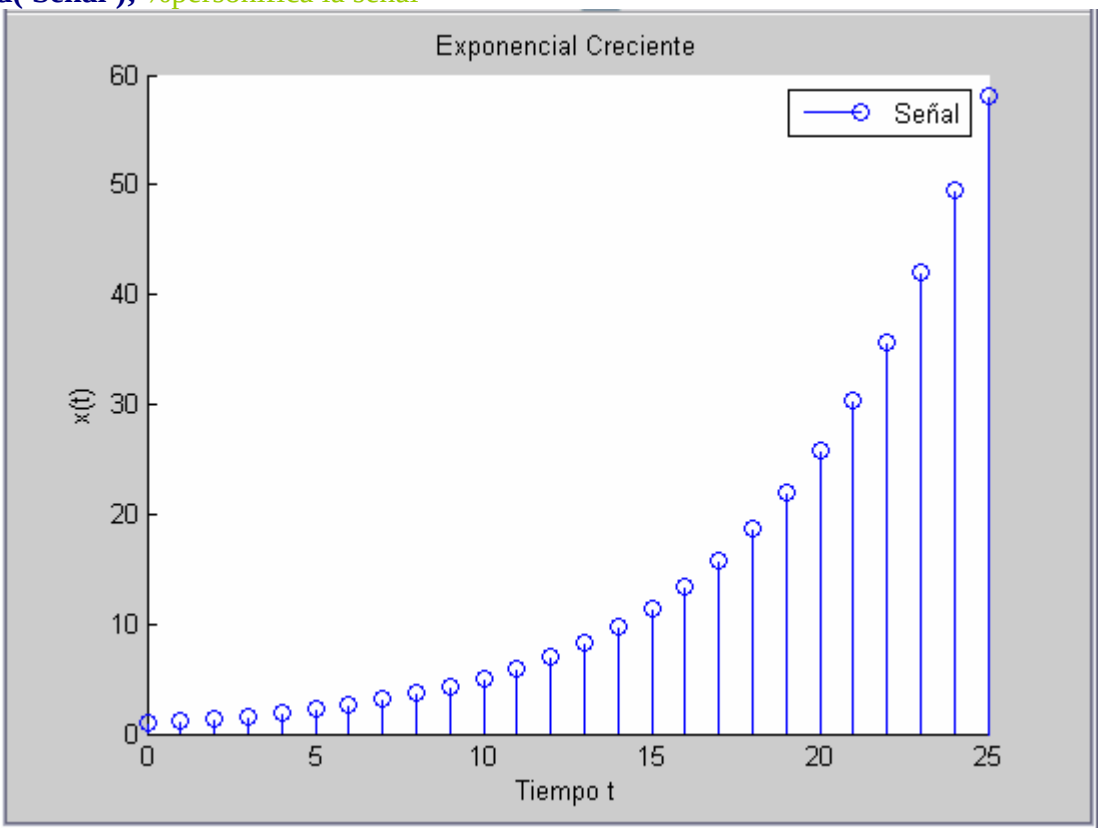
Para generar una Señal Exponencial Creciente en Tiempo Continuo, empleamos el siguiente conjunto completo de comandos:

```
>>B = 1;
>>a = 5;
>>t = 0:.001:1;
>>x = B*exp(a*t);
>>subplot (2,2,3) , plot(t,x,'r')  % Para Graficar el t y tri
>>title('Exponencial Creciente); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
```



Para generar una Señal Exponencial Creciente en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos:

```
>>B = 1;
>>r =0.85;
>>n = -0:25;
>>x = B*r.^-n;
>>subplot (2,2,4) , stem(n,x,'b')  % Para Graficar el t y tri
>>title('Exponencial Creciente); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
```



Práctica No. 3

- **Generación de Señales Senoidales**

MATLAB contiene también funciones trigonometricas que pueden usarse para generar señales senoidales. Una señal Coseno de amplitud **A**, frecuencia **w0**(medida en radianes por segundo) y ángulo de fase **phi** (en radianes) se obtiene empleando el comando:

```
>> A*cos(w0*t + phi);
```

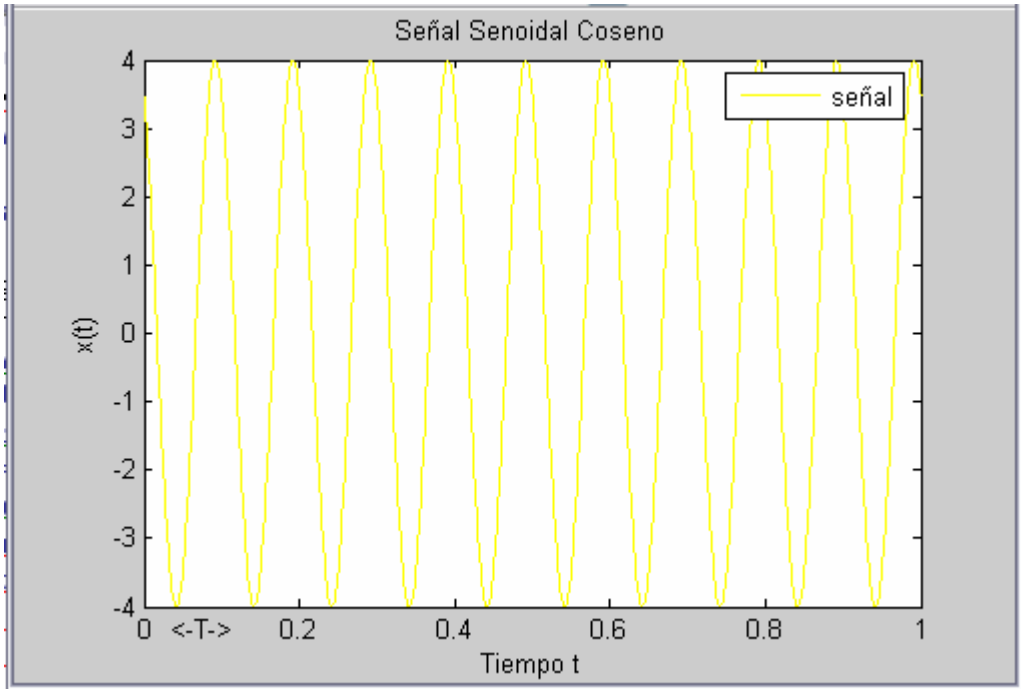
De modo alternativo, es posible utilizar la función seno para generar una señal senoidal empleando el comando.

```
>> A*sin(w0*t + phi);
```

Estos dos comandos se utilizan para generar las señales senoidales .

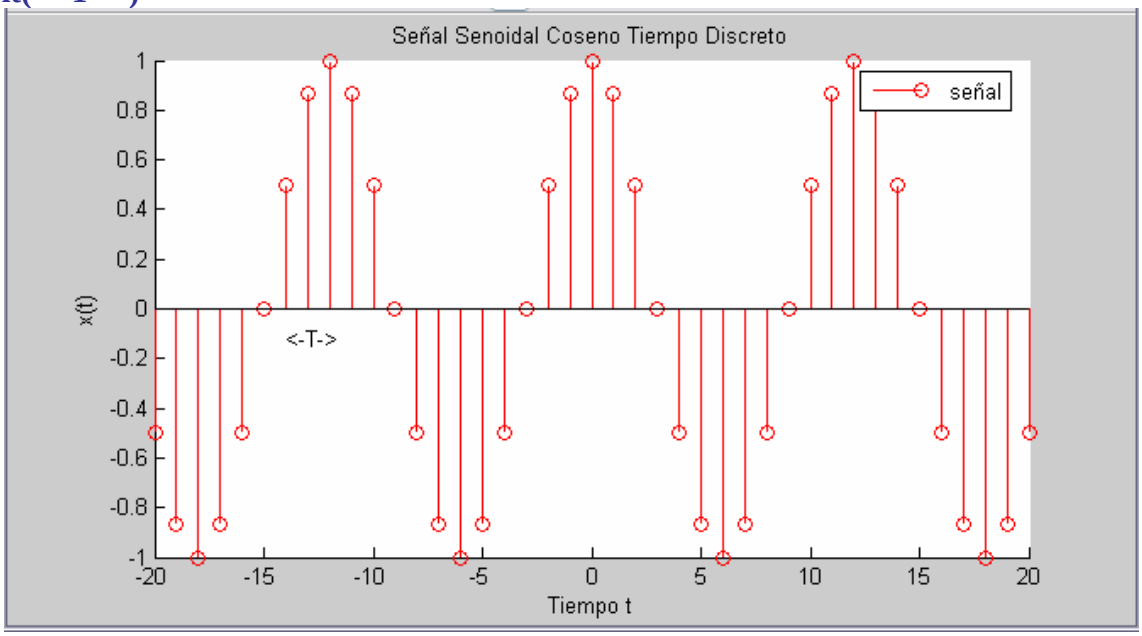
Para generar una Señal Senoidal para la señal coseno en Tiempo Continuo, empleamos el siguiente conjunto completo de comandos:

```
>>  A = 4;
>>w0 = 20*pi;
>> phi = pi/6;
>> t = 0:.001:1;
>> coseno = A*cos(w0*t + phi);
>> subplot (2,2,1) , plot(t,coseno,'y')  % Para Graficar el t y coseno
>> title('Señal Senoidal Coseno'); %Titulo de la hoja grafica
>> xlabel('Tiempo t'); %personifica el eje X
>> ylabel('x(t)'); %Personifica el eje Y
>> legend('señal'); %personifica la señal
>> gtext('<-T->')
```



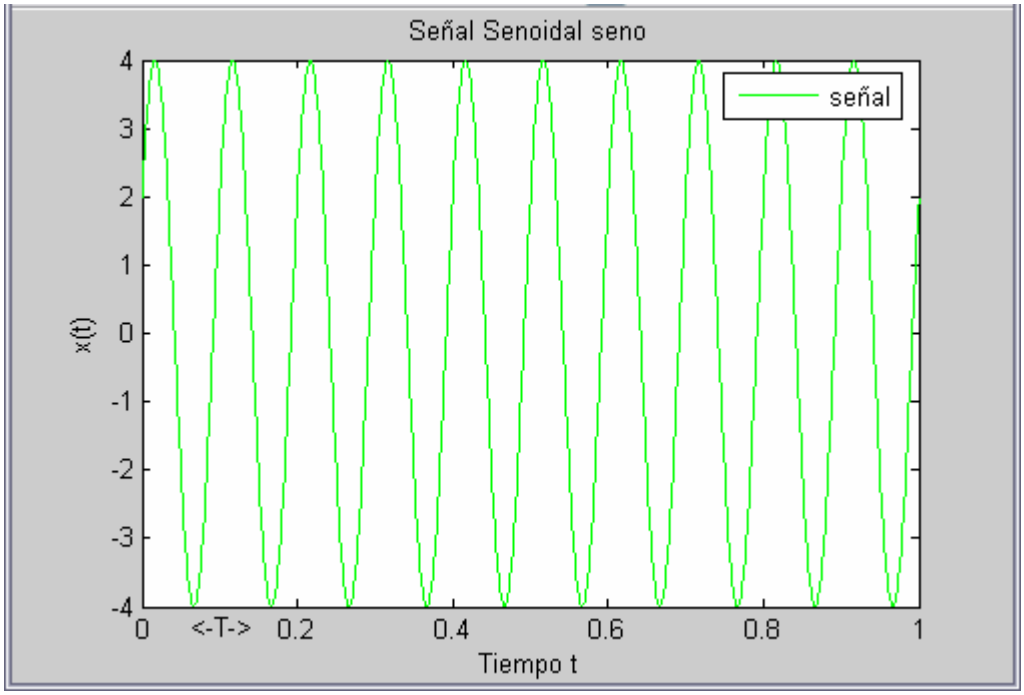
Para generar una Señal Senoidal para la señal coseno en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos:

```
>>A = 1;
>>omega = 2*pi/12; %frecuencia angular
 >>phi = 0;
 >>n = -20:20;
 >>coseno = A*cos(omega*n);
 >>subplot (2,2,2) , stem (n,coseno,'r')  % Para Graficar el n y coseno
 >>title('Señal Senoidal Coseno Tiempo Discreto'); %Titulo de la hoja grafica
 >>xlabel('Tiempo t'); %personifica el eje X
 >>ylabel('x(t)'); %Personifica el eje Y
 >>legend('señal'); %personifica la señal
 >>gtext('<-T->')
```



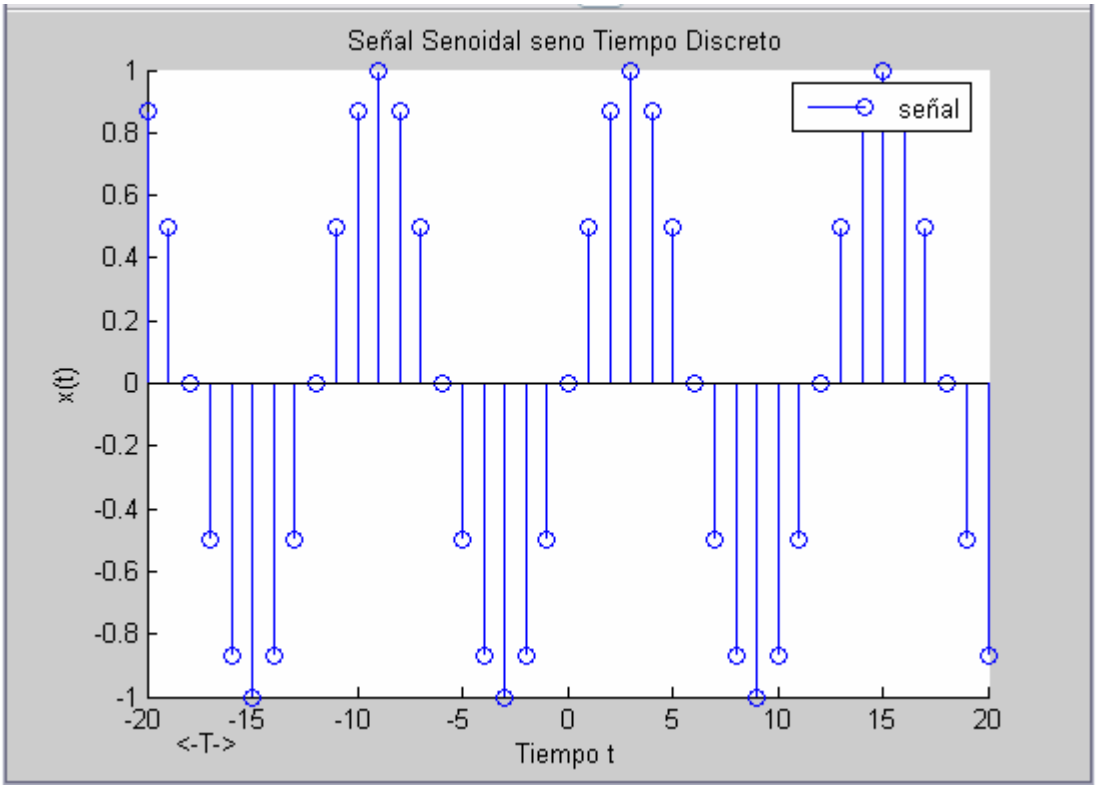
Para generar una Señal Senoidal para la señal seno en Tiempo Continuo, empleamos el siguiente conjunto completo de comandos:

```
>>  A = 4;
>>w0 = 20*pi;
>> phi = pi/6;
>> t = 0:.001:1;
>> seno = A*sin(w0*t + phi);
>> subplot (2,2,3) , plot(t,seno,'g')  % Para Graficar el t y coseno
>> title('Señal Senoidal seno'); %Titulo de la hoja grafica
>> xlabel('Tiempo t'); %personifica el eje X
>> ylabel('x(t)'); %Personifica el eje Y
>> legend('señal'); %personifica la señal
>> gtext('<-T->')
```



Para generar una Señal Senoidal para la señal seno en Tiempo Discreto, empleamos el siguiente conjunto completo de comandos

```
>>A = 1;
>>omega = 2*pi/12; %frecuencia angular
>>phi = 0;
>>n = -20:20;
>>seno = A*sin(omega*n);
>>subplot (2,2,4) , stem (n,seno,'b')  % Para Graficar el n y seno
>>title('Señal Senoidal seno Tiempo Discreto'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('señal'); %personifica la señal
>>gtext('<-T->')
```


Señal Senoidal seno Tiempo Discreto
señal
x(t)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-20
-15
-10
-5
0
5
10
15
20
<-T->
Tiempo t

Práctica No. 4

➢ Generación de Señales Senoidales Amortiguadas Exponencialmente

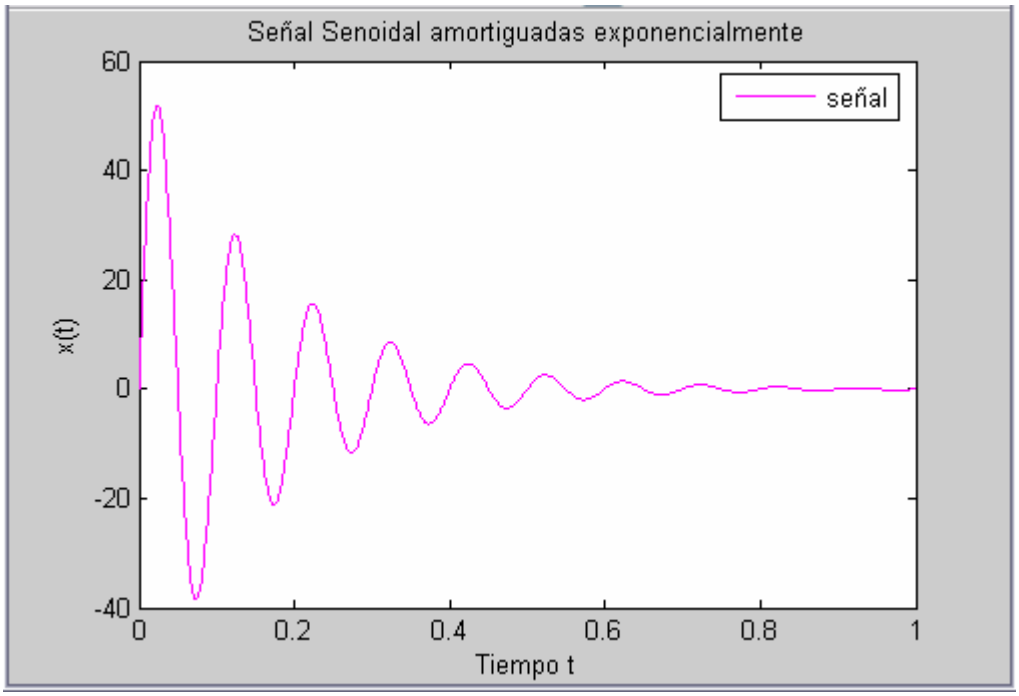
En las practicas anteriores, podemos ver que los comandos de generación de señales, se ha generado la amplitud deseada multiplicando un escalar, **A** en un vector que representa un señal de amplitud unitaria (por ejemplo, **sin(w0*t + phi))**. Esta operación se describe empleando un asterisco. Ahora consideremos la generación de una señal que requiere la **multiplicación elemento por elemento de dos vectores.**

Suponga que multiplicamos una señal senoidal por una señal exponencial para producir una señal senoidal amortiguada exponencialmente. Con cada componente de la señal siendo representado por un vector, la generación de tal señal producto requiere la multiplicación de un vector por otro vector n una base de elemento por elemento. MATLAB representa la multiplicación elemento por elemento utilizando un punto seguido por un asterisco. De tal modo, el comando para general la señal senoidal amortiguada exponencialmente.

A*sin (w0*t + phi).*exp(-a*t);

Un decaimiento exponencial, **a** es positiva, el conjunto completo de comandos es como sigue:

```
>>A = 60;
>>w0 = 20*pi; %frecuencia angular
>>phi = 0;
>>a = 6
>>t = 0:.001:1;
>>expseno = A*sin(w0*t + phi).*exp(-a*t);
>>subplot (2,2,1) , plot (t,expseno,'g')  % Para Graficar el t y coseno
>>title('Señal Senoidal amortiguadas exponencialmente'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('señal'); %personifica la señal
```



Práctica No. 5

- **Función Escalón, de Impulso y de Rampa**

En MATLAB, **ones (M , N)** es una matriz M x N de unos, y **zeros (M , N)** es una matriz de M x N de ceros. Podemos usar estas matrices para generar dos señales empleadas comúnmente, del siguiente modo:

- **Función Escalón**. Una función escalón de magnitud unitaria se genera al escribir.

u = [zeros (1,50), ones(1,50)];

- **Función en tiempo discreto**: Un impulso en tiempo discreto de magnitud unitaria se genera escribiendo.

Delta = [zeros (1,49), 1, zeros(1, 49)];

Para generar una secuencia de la rampa, simplemente escribimos

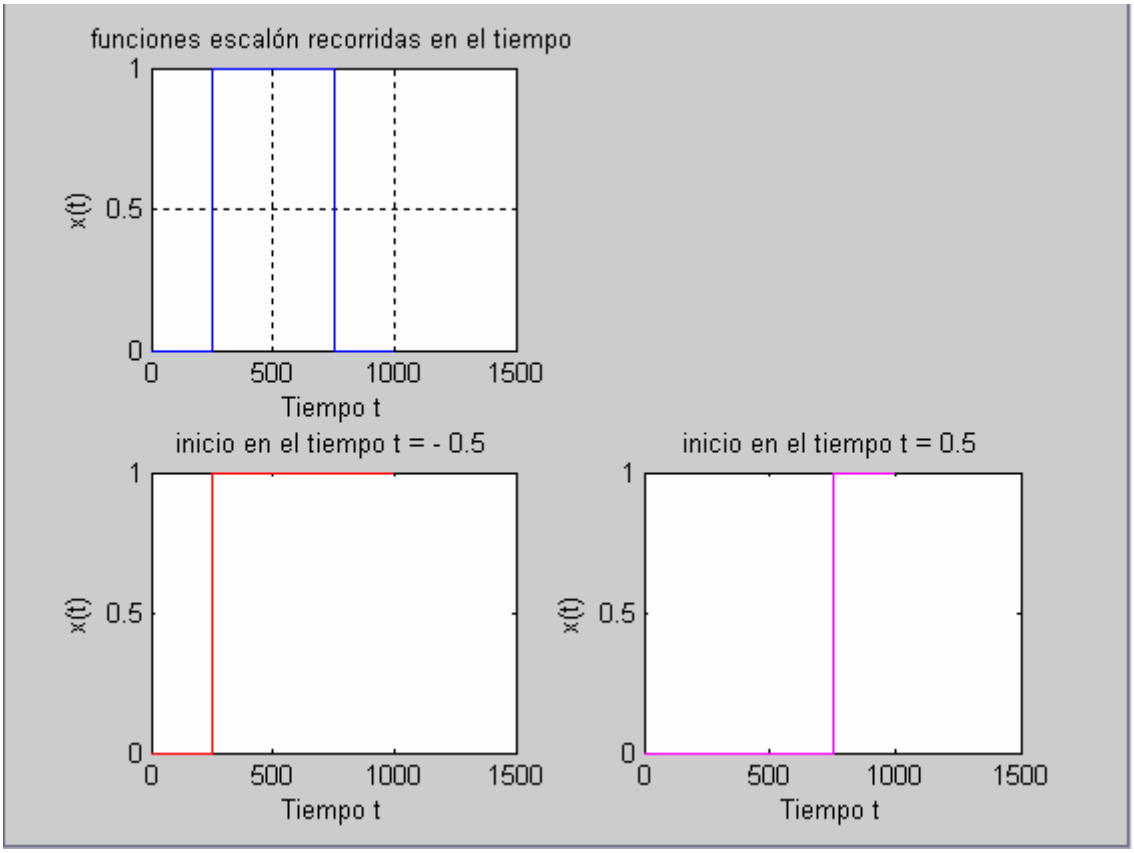
Ramp = n;

Ejercicio.

Ilustrar como un par de funciones escalón recorridas en el tiempo, una con relación a la otra, pueden utilizarse para producir un pulso rectangular.

Reps:

```
>>t= -1:1/500:1; %define los valores de tiempo de -1 seg a 1 seg
>>u1 = [zeros(1,250), ones(1,751)]; %Genera función escalón de amplitud unitaria, con inicio en el tiempo t=-0.5
>>u2 = [zeros(1,751), ones(1,250)]; %Genera una segunda función escalón de amplitud unitaria con inicio en t = 0.5 seg
>>u = (u1) - (u2); %Sustrae U2 de u1 para producir un pulso rectangular de amplitud y duración unitarias centrado en el origen
>>subplot (2,2,1), plot (u,'b')
>>title('funciones escalón recorridas en el tiempo'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>hold on
>>subplot (2,2,3), plot (u1,'r')
>>title('inicio en el tiempo t = - 0.5'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>subplot (2,2,4), plot (u2,'m')
>>title('inicio en el tiempo t = 0.5'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>hold off
```


funciones escalón recorridas en el tiempo
x(t)
Tiempo t
inicio en el tiempo t = - 0.5
x(t)
Tiempo t
inicio en el tiempo t = 0.5
x(t)
Tiempo t

Práctica No. 6

Operaciones Básicas Sobre Señales

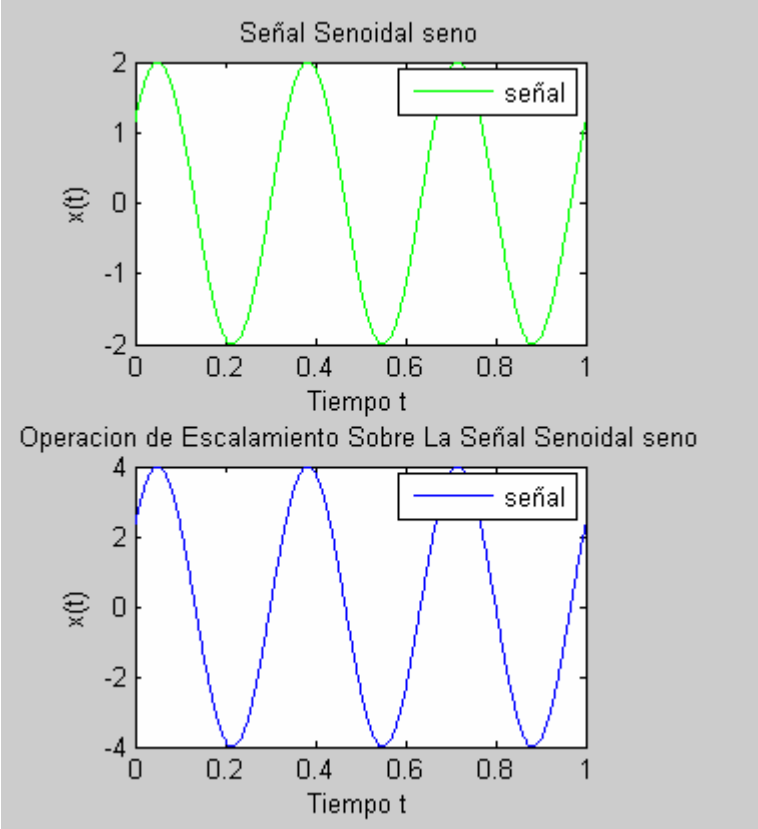
Operaciones realizadas sobre variables dependientes

- **Operacion de Escalamiento:** Es multiplicar una función por un escalar o (factor de escala) ¨**c**¨. Esto equivale al efecto fisico del amplificador, donde el facto de escala ¨**c**¨ es la **Ganancia**.
  - En **Tiempo Continuo** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = C * X_{(t)}}$
  - En **Tiempo Discreto** se expresa de la siguiente manera: $\mathbf{Y_{[n]} = C * X_{[n]}}$

  En este caso generaremos una **Señal Senoidal** para ejemplificar la operación de escalamiento. Donde multiplicaremos la función **F(t),** en este caso nuestra función es **seno = A*sin(w0*t + phi),** la cual será multiplicada por el factor de escala **C** el cual tendrá un valor de **2**. Aquí veremos como la primera señal generada tiene una amplitud de 2. La cual al aplicar la operación de escalamiento, multiplicada por dos en este caso, tendrá una amplitud de 4.

Para generar una Señal Senoidal para la señal seno en Tiempo Continuo, aplicándole la operación de escalamiento, empleamos el siguiente conjunto completo de comandos: El mismo procedimiento utilizaríamos en caso del tiempo discreto.

```
>> A = 2; %Amplitud de la señal Senoidal Original
>> w0 = 6*pi; %Frecuencia Fundamental
>> phi = pi/5;
>> t = 0:.001:1; %Intervalo de muestreo de F de 1ms
>> seno = A*sin(w0*t + phi); %Función para generar la función seno
>> hold on
>> subplot (2,2,1), plot(t,seno,'g')  % Para Graficar el t y coseno
>> title('Señal Senoidal seno'); %Titulo de la hoja grafica
>> xlabel('Tiempo t'); %personifica el eje X
>> ylabel('x(t)'); %Personifica el eje Y
>> legend('señal'); %personifica la señal
>> x = seno * 2  %factor de escala ¨c¨= 2. En este caso.
>> hold on
>> subplot (2,2,3), plot(t,x,'b')
>> title('Operacion de Escalamiento Sobre La Señal Senoidal seno'); %Titulo de la hoja grafica
>> xlabel('Tiempo t'); %personifica el eje X
>> ylabel('x(t)'); %Personifica el eje Y
>> legend('señal'); %personifica la señal
>> hold off
```



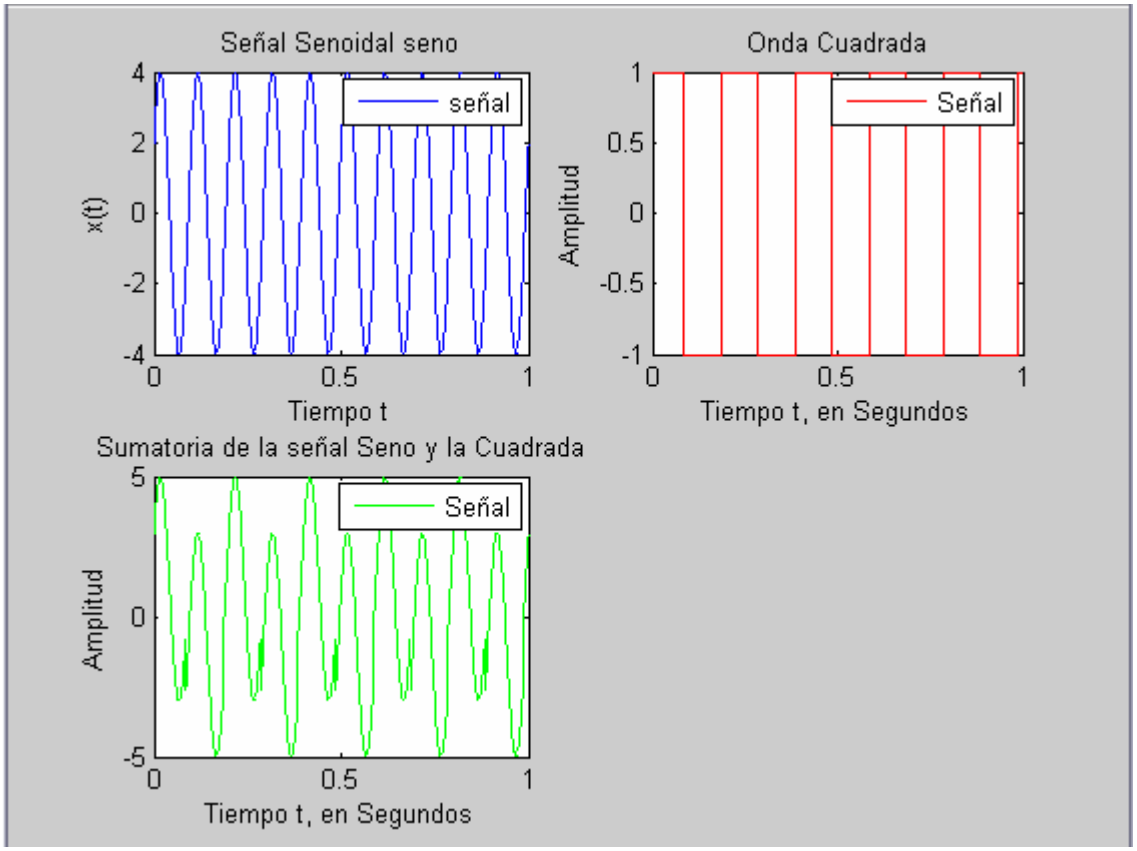
Ejercicio.....

- **<u>Operacion de Suma:</u>** Consiste en sumar ($\overset{+}{-}$) dos señales. Esta funcion equivale a la funcion que realiza un dispositivo o **Etapa Mezcladora**.
  - En **Tiempo Continuo** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = X_{1(t)} \overset{+}{-} X_{2(t)}}$
  - En **Tiempo Discreto** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = X_{1[n]} \overset{+}{-} X_{2[n]}}$

  Ejemplo.

```
>>a = 4;
>>w0 = 20*pi;
>>phi = pi/6;
>>t = 0:.001:1;% Intervalo de muestreo F   de 1 ms sobre el intervalo de 0 a 1s
>>seno = a*sin(w0*t + phi);
>>subplot (2,2,1) , plot(t,seno,'b')  % Para Graficar el t y coseno
>>hold on
>>title('Señal Senoidal seno'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('señal'); %personifica la señal

>>A = 1;  %Representa la Amplitud que en este caso es de 1
>>W0 = 10*pi; %Frecuencia Fundamental
>>rho = 0.5; % La fracción de cada periodo en el cual la señal es positive.
>>sq = A*square(W0*t + rho);   % Para generar la Onda Cuadrada
>>subplot (2,2,2), plot(t,sq,'r');  % Para Graficar el t (intervalo de muestreo)
>>hold on
>>title('Onda Cuadrada'); %Titulo de la hoja grafica
```

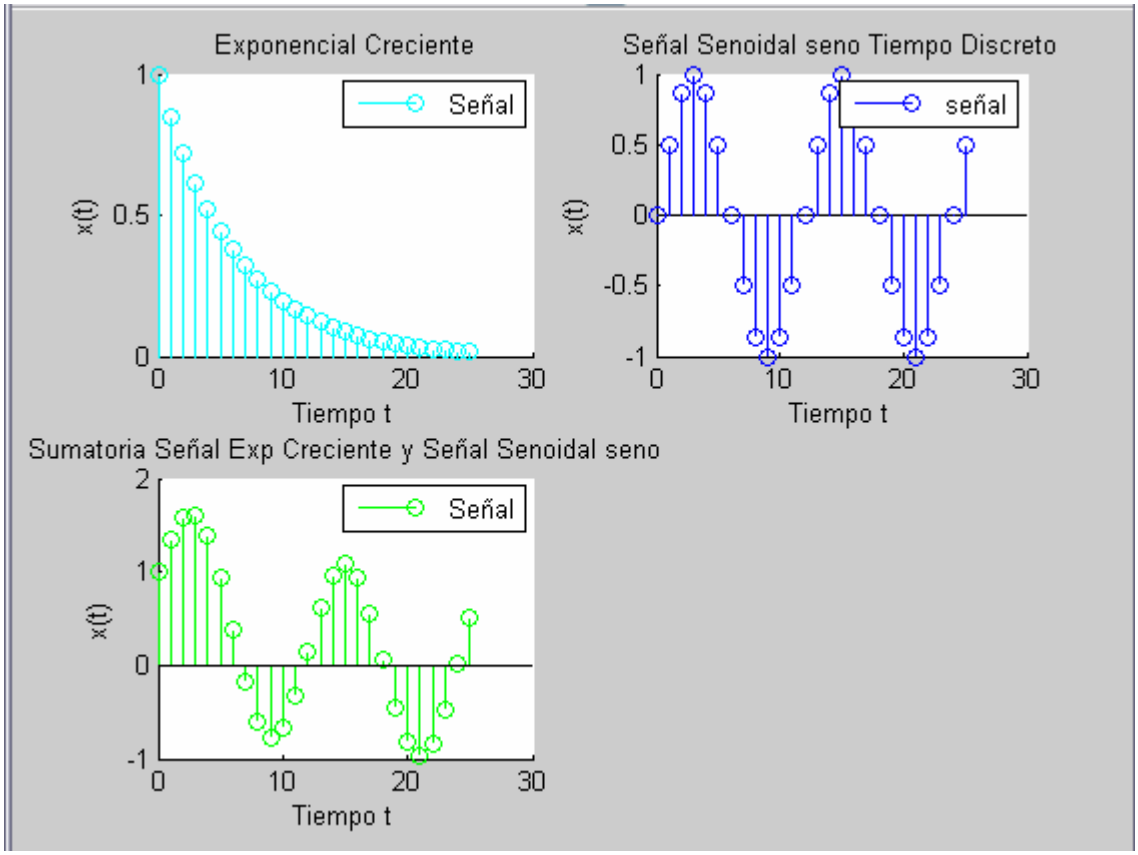
```
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal

>>X = seno + sq; % Suma la señal Senoidal a la señal Cuadrada
>>subplot (2,2,3), plot(t,X,'g')
>>hold on
>>title('Sumatoria de la señal Seno y la Cuadrada'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('Amplitud'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>hold off
```



Ejercicio.....

```
>>A = 1;
>>r = 0.85;
>>n = -0:25;
>>h = A*r.^n;
>>Subplot (2,2,1), stem(n,h,'c')  % Para Graficar el t y tri
>>hold on
>>title('Exponencial Creciente'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>omega = 2*pi/12; %frecuencia angular
>>phi = 0;
```

```
>>seno = A*sin(omega*n);
>>subplot (2,2,2) , stem(n,seno,'b')  % Para Graficar el n y seno
>>hold on
>>title('Señal Senoidal seno Tiempo Discreto'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('señal'); %personifica la señal
>>X = h + seno;
>>subplot (2,2,3), stem(n,X,'g'); %Su de la señal Exponencial Creciente y de Decreciente
>>hold on
>>title('Sumatoria Señal Exp Creciente y Exp Decreciente'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>hold off
```



- **Operacion de Multiplicacion:** Consiste en multiplicar (*) dos señales. Esta operacion en la practica la realiza lo que conocemos como **Modulacion**.

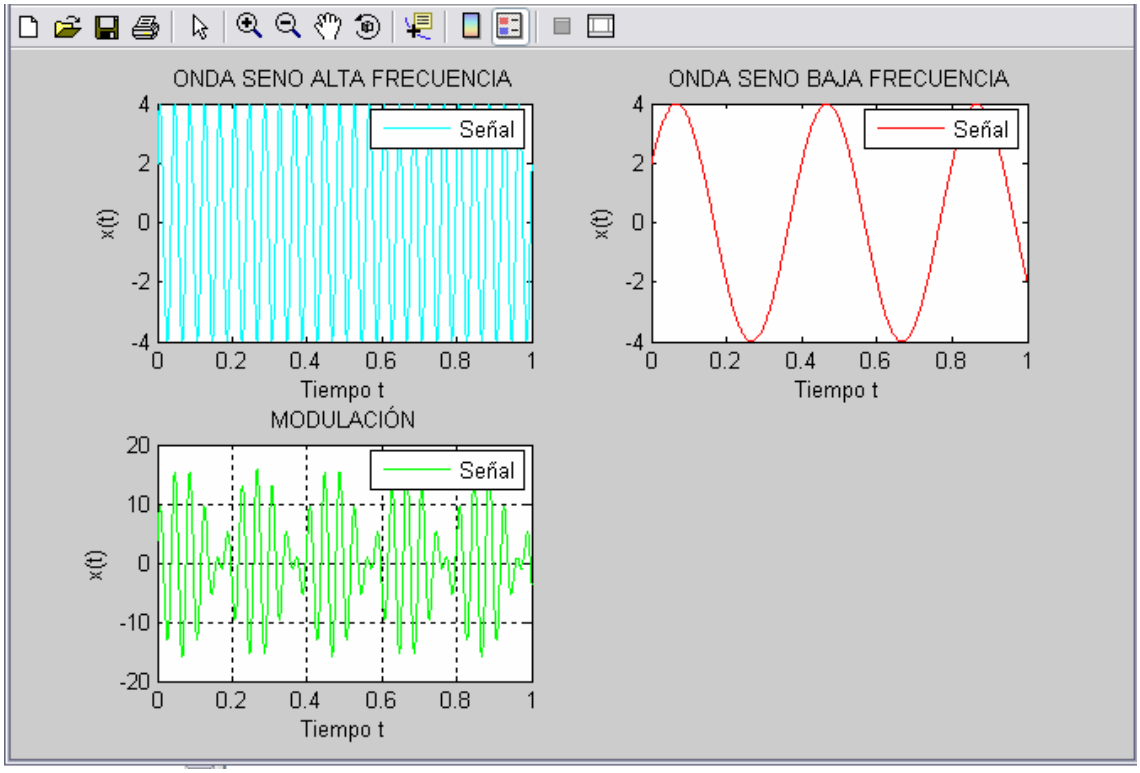
- En **Tiempo Continuo** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = X_{1(t)} * X_{2(t)}}$
- En **Tiempo Discreto** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = X_{1[n]} * X_{2[n]}}$

Ejemplo

MULTIPLICACIÓN DE DOS SEÑALES

En este ejercicio generaremos dos señales Seno, una de alta frecuencia y otra de baja frecuencia y luego multiplicaremos resultados de dichas señales. Produciendo de esta manera la modulacion.

```
% PRIMERA SEÑAL
>>t = 0:.001:1;%Vector tiempo
>>A= 4; %La amplitud;
>>W = 50*pi; %La frecuenia;
>>PHI = pi/6;% ángulo de desfase;
>>SENO1 = A*sin(W*t + PHI);
>>Subplot (2,2,1), plot(t,SENO1,'c')  % Para Graficar el t y tri
>>hold on
>>title('ONDA SENO ALTA FRECUENCIA'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal

% SEGUNDA SEÑAL
>>A = 4; %La amplitud;
>>W = 5*pi; %La frecuenia;
>>PHI = pi/6; % ángulo de desfase;
>>SENO2 = A*sin(W*t + PHI);
>>Subplot (2,2,2), plot(t,SENO2,'r')  % Para Graficar el t y tri
>>hold on
>>title('ONDA SENO BAJA FRECUENCIA'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal

%MODULACIÓN
>>MULT = SENO1 .* SENO2;
>>Subplot (2,2,3), plot(t,MULT,'g')  % Para Graficar el t y tri
>>hold on
>>title('MODULACIÓN'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>hold off
>>grid on
```



- **<u>Operacion de Diferenciacion:</u>** Esta operacion solo se realiza en **Tiempo Continuo.** El dispositivo fisico que efectua esta operacion es el **Inductor o Bobina.**

- En **Tiempo Continuo** se expresa de la siguiente manera: $\mathbf{Y_{(t)} = d/d_{(t)} * X_{(t)}}$

Ejemplo

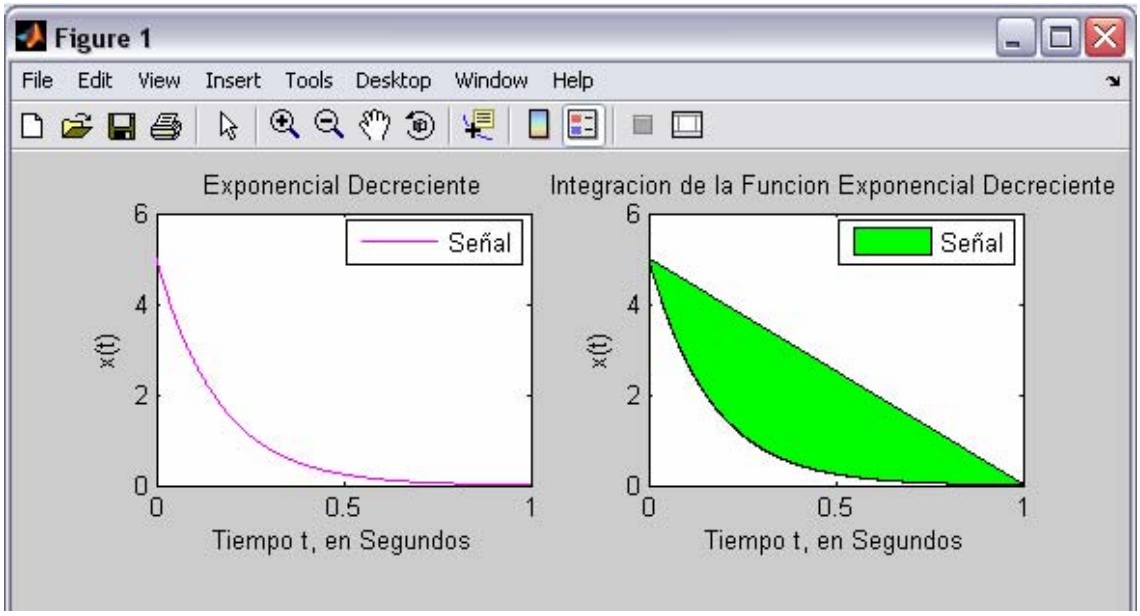
- **<u>Operacion de Integracion:</u>** Esta operacion solo se realiza en **Tiempo Continuo.** El dispositivo fisico que efectua esta operacion es el **Capacitor o Condensador.**
  - En **Tiempo Continuo** se expresa de la siguiente manera:

$$Y(t) = \int_{-\infty}^{t} X(t)\, dt$$

Ejemplo
Generaremos una Función Exponencial Decreciente, y luego a dicho resultado le aplicaremos la Integración.

```
>>B = 5;
>>a = 6;%Amplitud
>>t = 0:.001:1;%Vector de Tiempo
>>x = B*exp(-a*t); %Para generar la señal Exponencial Decreciente;
>>subplot (2,2,1) , plot(t,x,'m')  % Para Graficar el t y tri
>>title('Exponencial Decreciente'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>subplot (2,2,2), fill(t,x,'g') % Integra la funcion Exponencial y luego grafica la señal.
>>title('Integracion de la Funcion Exponencial Decreciente'); %Titulo de la hoja grafica
```

```
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
```



Ejemplo 2
En esta ocacion generaremos una Onda Coseno, y luego la Integramos.

```
% Generación de onda Coseno.

>>t = 0:.001:1; %Vector de tiempo
>>A = 4; %La amplitud;
>>W = 20*pi;%La frecuenia;
>>PHI = pi/6;%PHI ángulo de desfase;
>>COSENO = A*cos(W*t + PHI);

%INTEGRAL

>>subplot(2,2,1), plot (t,COSENO,'g')
>>hold on
>>title('Onda Coseno'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal

>>subplot(2,2,2), fill(t,COSENO,'b');
>>title('INTEGRACIÓN DEL COSENO'); %Titulo de la hoja grafica
>>xlabel('Tiempo t, en Segundos'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Señal'); %personifica la señal
>>hold off
```

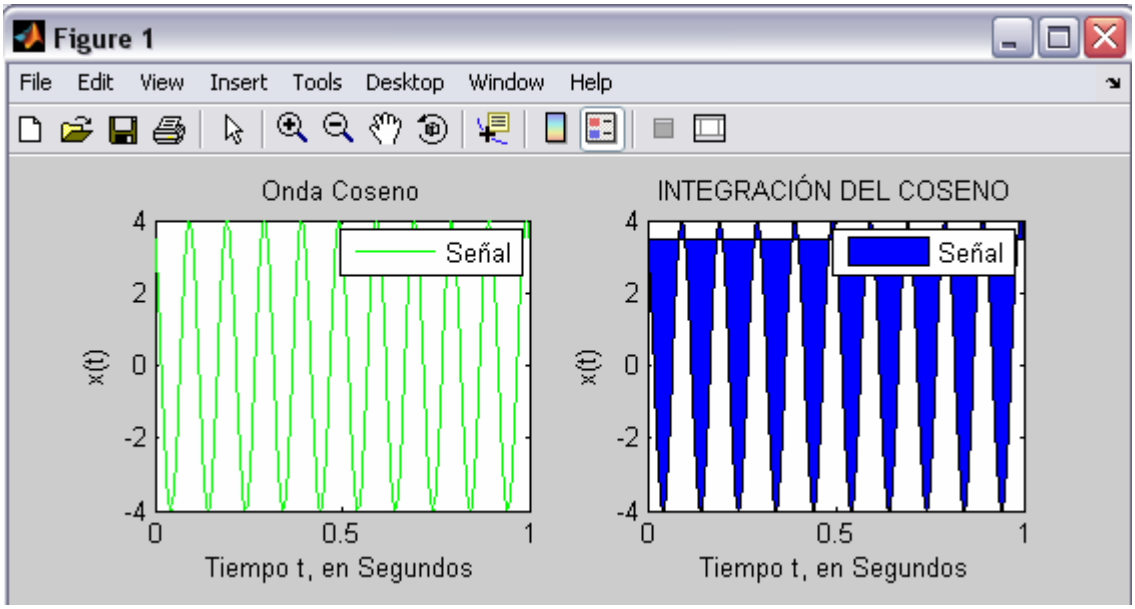


Figure 1
File Edit View Insert Tools Desktop Window Help
Onda Coseno
Señal
x(t)
Tiempo t, en Segundos
INTEGRACIÓN DEL COSENO
Señal
x(t)
Tiempo t, en Segundos

Práctica No. 7

Operaciones realizadas sobre variables independientes

- **Operacion De Escalamiento En El Tiempo:**

- Sea $\mathbf{X_{(t)}}$ una Señal en **Tiempo Continuo.** La Señal $\mathbf{Y_{(t)}}$ obtenida por el escalamiento de la variable independiente, tiempo **t** por un factor **a**, es decir, cuando se multiplica la variable independiente por un factor `**a**`, se define como: $\mathbf{Y_{(t)} = X (a * t)}$

Dado esto decimos lo siguiente:
-Si $\mathbf{a > 1}$ la señal $\mathbf{Y_{(t)}}$ es una version **comprimida** de $\mathbf{X_{(t)}}$.
-Si $\mathbf{0 < a < 1}$ ò $\mathbf{a < 1}$ la señal $\mathbf{Y_{(t)}}$ es una version **expandida** de $\mathbf{X_{(t)}}$.

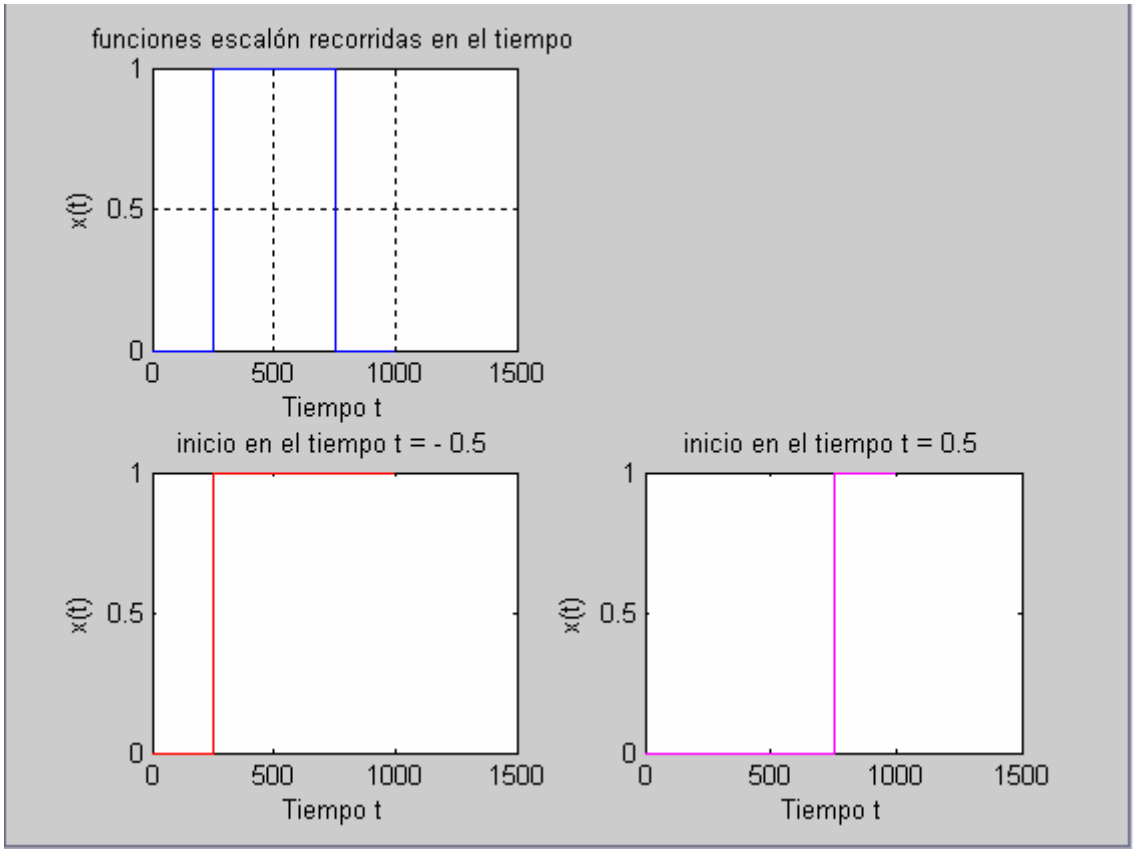
- Sea $\mathbf{X_{[n]}}$ una Señal en **Tiempo Discreto.** La Señal $\mathbf{Y_{[n]}}$ obtenida por el escalamiento de la variable independiente, tiempo **n** por un factor **k**, es decir, cuando se multiplica la variable independiente por un factor `**k**`, se define como: $\mathbf{Y_{[n]} = X (k * n)}$
Esta solo puede **comprimirse** pero no **expandirse.**

Dado esto decimos lo siguiente:
-Si $\mathbf{k > 0}$ la señal $\mathbf{Y_{[n]}}$, solo se define para valores enteros de **k.**
-Si $\mathbf{k > 1}$ entonces algunos valores de la señal en tiempo discreto $\mathbf{Y_{[n]}}$ se pierden.

Ilustrar como un par de funciones escalón recorridas en el tiempo, una con relación a la otra, pueden utilizarse para producir un pulso rectangular.

Ejemplo

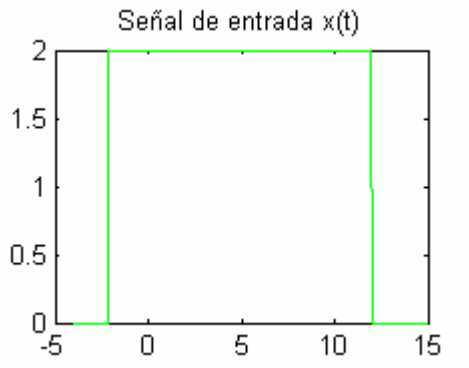
```
>>t= -1:1/500:1; %define los valores de tiempo de -1 seg a 1 seg
>>u1 = [zeros(1,250), ones(1,751)]; %Genera función escalón de amplitud unitaria, con inicio en el tiempo t=-0.5
>>u2 = [zeros(1,751), ones(1,250)]; %Genera una segunda función escalón de amplitud unitaria con inicio en t = 0.5 seg
>>u = (u1) - (u2); %Sustrae U2 de u1 para producir un pulso rectangular de amplitud y duración unitarias centrado en el origen
>>subplot (2,2,1), plot (u,'b')
>>title('funciones escalón recorridas en el tiempo'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>hold on
>>subplot (2,2,3), plot (u1,'r')
>>title('inicio en el tiempo t = - 0.5'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>subplot (2,2,4), plot (u2,'m')
>>title('inicio en el tiempo t = 0.5'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>hold off
```



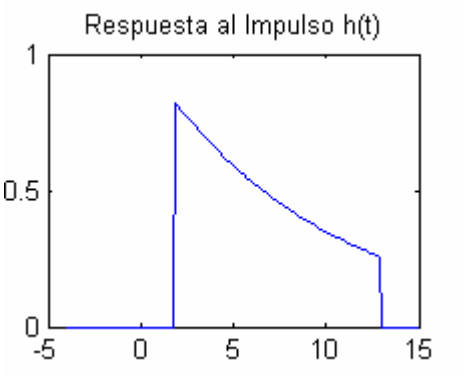
CONVOLUCIÓN

****

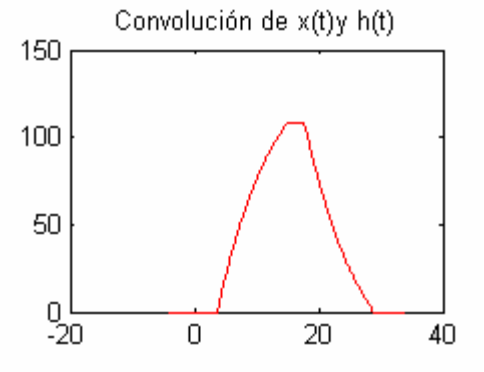
```
%Convolución en tiempo continuo
tc = -4:0.1:15;
xt = 2 .* ones(1,191);
xt(1:19) = 0;
xt(161:191) = 0;
ht = 0.9 .^tc;
ht(1:59) = 0;
ht(171:191)= 0;
ytconv = conv(xt,ht);
tf = -4:0.1:34 ;

%Impresión convolución continua
subplot(3,3,1);
plot(tc,xt);
title('Señal de entrada x(t)');
```



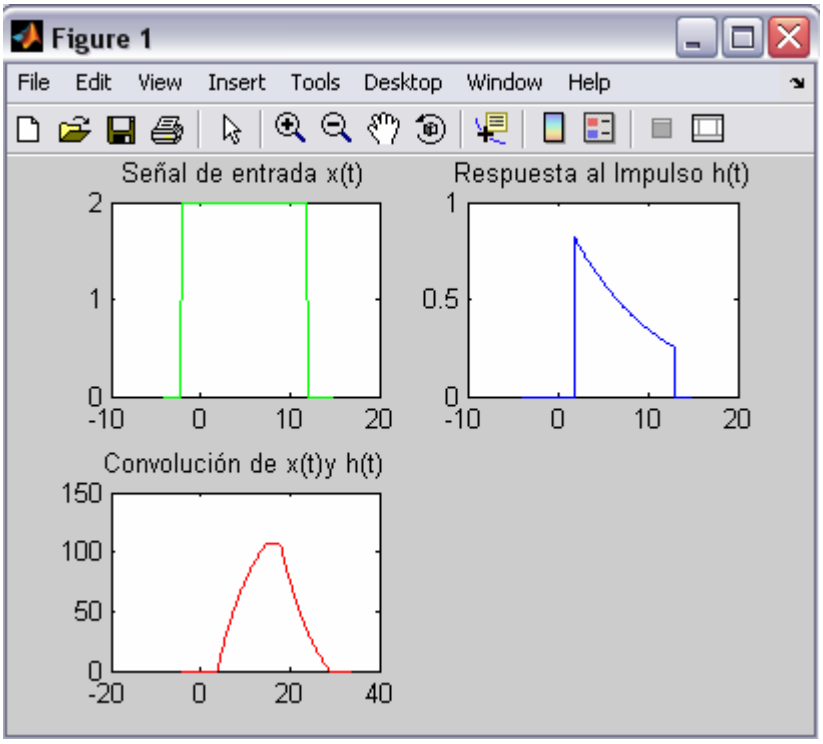

```
subplot(3,3,2);
plot(tc,ht);
title('Respuesta al Impulso h(t)');
```



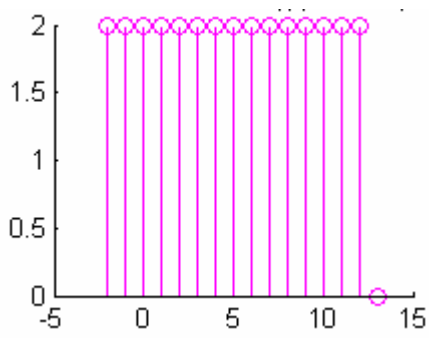

```
subplot(3,3,3);
plot(tf,ytconv);
title('Convolución de x(t)y h(t)');
```



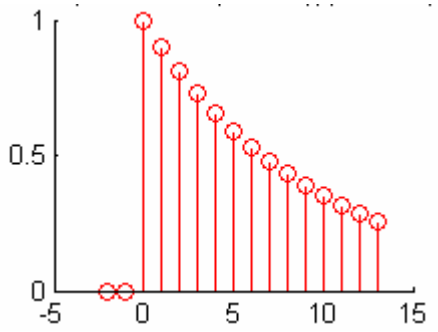
Representación Final.



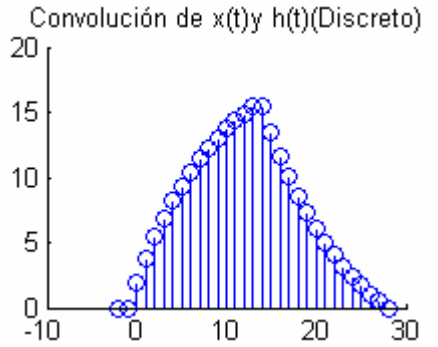

```
%Convolución en tiempo discreto
n = -2:13;
xn = 2 .* ones(1,16);
xn(16) = 0;
hn = 0.9 .^ n;
hn (1:2) = 0;
ynconv = conv(xn,hn);
nf = -2:28;

%impresión convolución discreta

subplot(3,3,4);
stem(n,xn);
title('Señal de entrada x(t)(Discreto)');
```



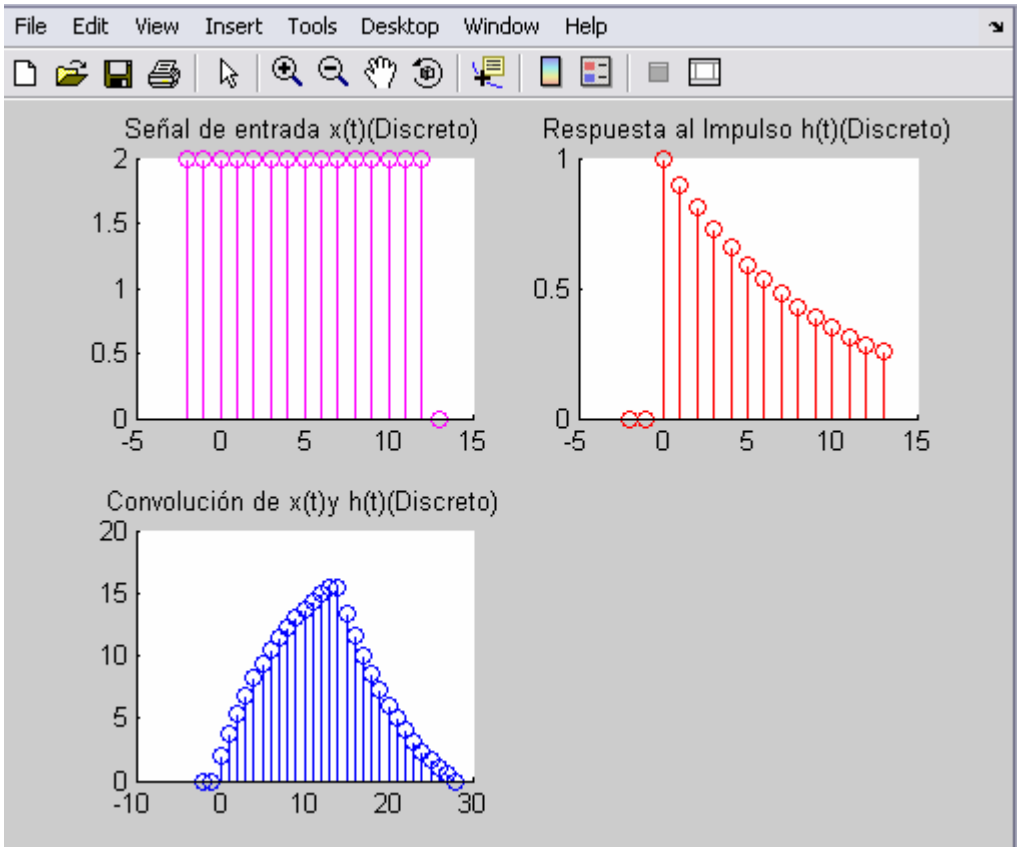
```
subplot(3,3,5);
stem(n,hn);
title('Respuesta al Impulso h(t)(Discreto)');
```



subplot(3,3,6);
stem(nf,ynconv);
title('Convolución de x(t)y h(t)(Discreto)');



Representación Grafica Final.



Práctica No. 8

FOURIER

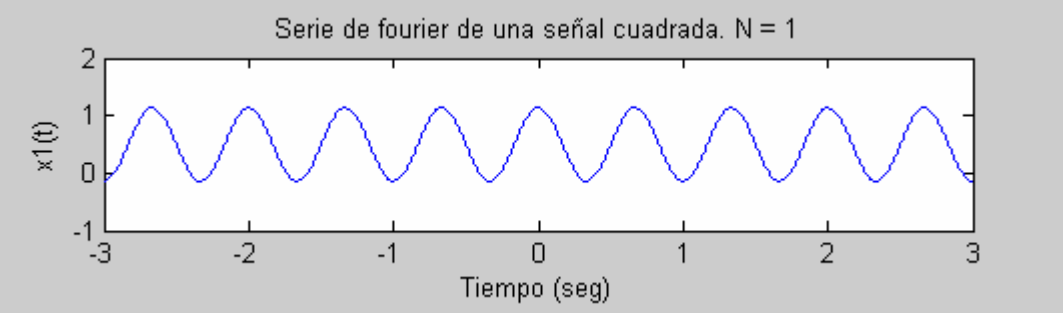
Este ejercicio practico, tiene la particularidad de por medio de la función imput, le permite introducir un valor (numero) para ingresar en el programa el numero de harmónicos para la serie de fourier.

En el siguiente ejemplo introduciremos varios valores para representar los harmonicos.

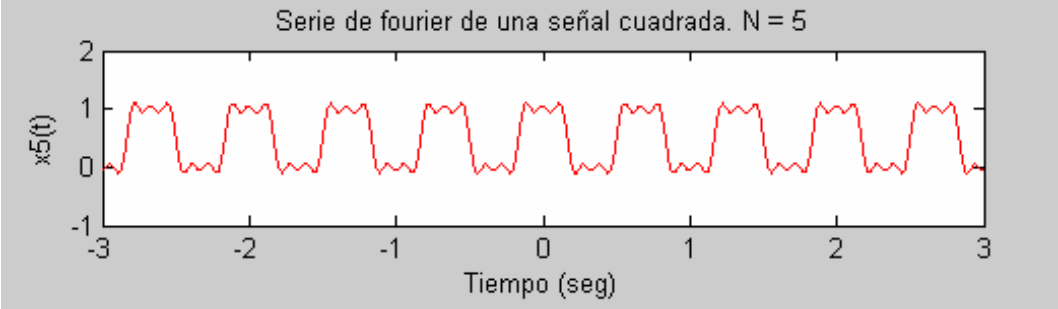
1) En el primer ejemplo introduciremos el valor o numero 1.
2) En el segundo ejemplo introduciremos el valor o numero 5.
3) En el tercer ejemplo introduciremos el valor o numero 10.

```
%Desarrollo de la serie de Fourier
t = -3:6/10000:3;
N = input('Números de harmónicos para la serie de fourier: '); %Por medio de este Compando de Imput le introducimos un Harmonico a la serie de fourier.
c0 = 0.5;
w0 = 3*pi;
xN = c0*ones(1,length(t));
for k=1:N,
  ck = 1/k/pi*sin(k*pi/2); %Coeficiente de Fourier
  c_k = ck;
  xN = xN + ck*exp(j*k*w0*t) + c_k*exp(-j*k*w0*t);
end

%Impresión serie de fourier
Subplot(3,1,3);
plot(t,xN)
title(['Serie de fourier de una señal cuadrada. N = ',num2str(N)])
xlabel('Tiempo (seg)')
ylabel(['x',num2str(N),'(t)'])
```

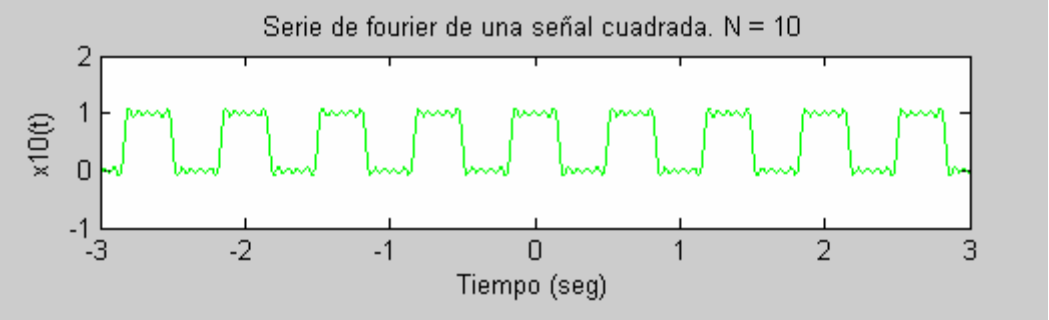
Ejemplo 1:



Ejemplo 2:



Ejemplo 3:



Práctica No. 9

- **Modulación Análoga**

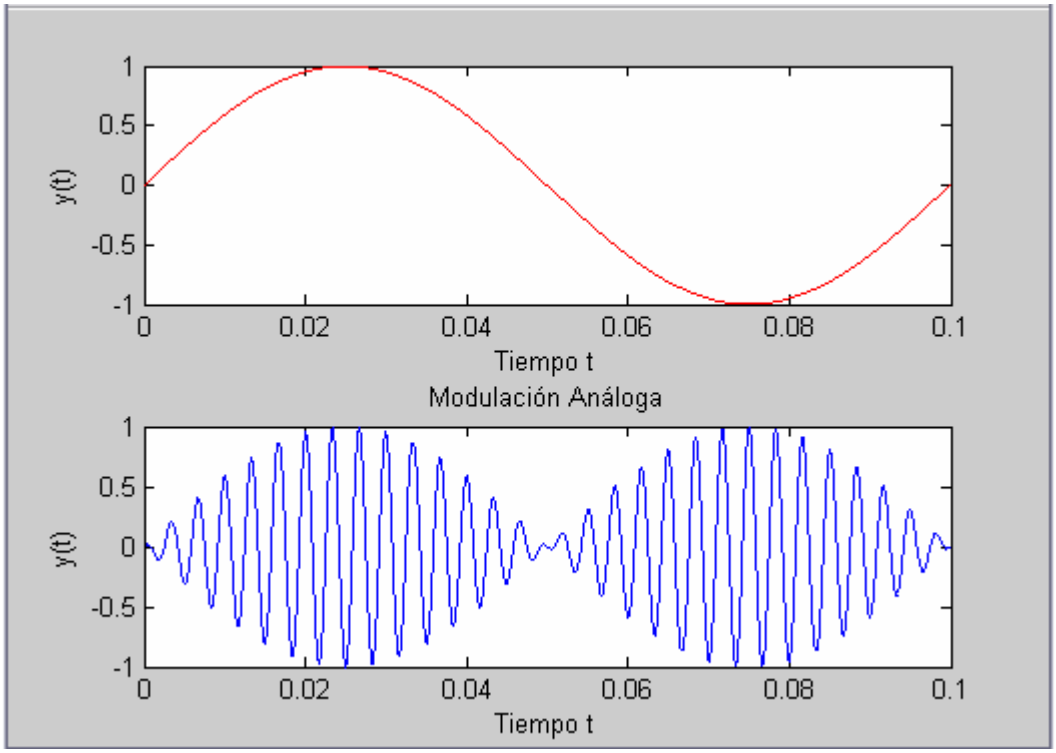
En esta práctica se describe como representar una señal análoga utilizando vectores o matrices. En esta práctica se proporcionan ejemplos utilizando tanto las funciones de modulación análoga como la demodulacion.

>Representación e la señal análoga

Para modular una señal análoga utilizando las herramientas, inicie con una verdadera señal de mensaje y una muestra es rate Fs en Hertz. Represente la señal que usa un vector x, las entradas de cual dan los valores de la señal en los incrementos de tiempo de 1/Fs. O bien, usted puede usar una matriz para representar una señal de varios canales, donde cada columna de la matriz representa un canal.

Por ejemplo, si la **t** mide el tiempo en segundos, entonces el vector **x** debajo es el resultado de probar una onda de seno 8000 veces por segundo durante 0.1 segundos. El vector y representa la señal modulada.
Para representar esto utilizamos el siguiente conjunto de comandos:

```
>>Fs = 8000; % El probar la razon es 8000 muestras por segundo.
>>Fc = 300; % Frecuencia Portadora en Hz
>>t = [0:.1*Fs]'/Fs; % Sampling times for .1 second
>>x = sin(20*pi*t); % Representación de la señal
>>y = ammod(x,Fc,Fs); % Modulación x que produce y.
figure;
>>title('Señal Análoga'); %Titulo de la hoja grafica
>>subplot(2,1,1); plot(t,x,'r'); % Grafique x 1era grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('y(t)'); %Personifica el eje Y
>>subplot(2,1,2); plot(t,y,'b') % Grafica y 2da grafica
>>title('Modulación Análoga'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('y(t)'); %Personifica el eje Y
```


1
0.5
0
-0.5
-1
y(t)
0
0.02
0.04
0.06
0.08
0.1
Tiempo t
Modulación Análoga
1
0.5
0
-0.5
-1
y(t)
0
0.02
0.04
0.06
0.08
0.1
Tiempo t

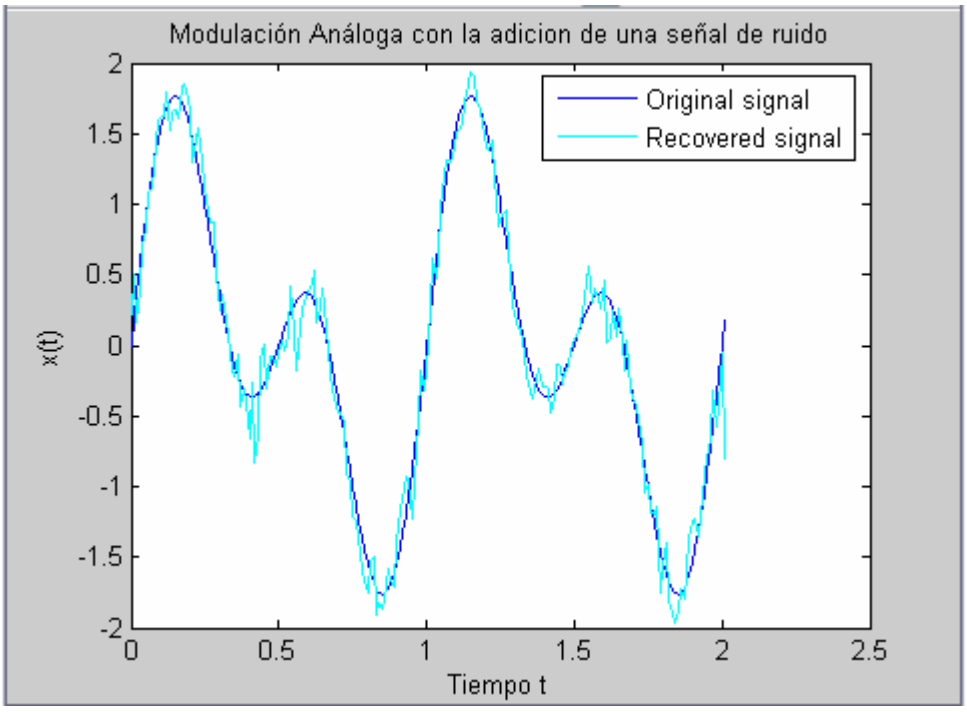
Práctica No. 10

- **Ejemplo de Modulación Análoga**

Este ejemplo ilustra el formato básico de la modulación análoga y funciones de desmodulación. Aunque la modulación de fase de empleos de ejemplo, la mayor parte de elementos de este ejemplo se aplique a otras técnicas de modulación análogas también.

El ejemplo prueba una señal analógica y lo modula. Entonces esto simula un ruido aditivo blanco Gaussiano (AWGN) el canal, desmodula la señal recibida, y traza el original y señales desmoduladas.

```
% Prepare to sample a signal for two seconds,
% at a rate of 100 samples per second.
>>Fs = 100;
>>t = [0:2*Fs+1]'/Fs;
>>x = sin(2*pi*t) + sin(4*pi*t);% Crear la señal, una suma senoidal.
>>Fc = 10; % Frecuencia portadora en modulación
>>phasedev = pi/2; % Desviación de fase para modulación de fase
>>y = pmmod(x,Fc,Fs,phasedev); % Modulación.
>>y = awgn(y,10,'measured',103); % Adición de ruido.
>>z = pmdemod(y,Fc,Fs,phasedev); % Desmodulación.
>>figure; plot(t,x,'b-',t,z,'c-');
>>title('Modulación Análoga con la adición de una señal de ruido'); %Titulo de la hoja grafica
>>xlabel('Tiempo t'); %personifica el eje X
>>ylabel('x(t)'); %Personifica el eje Y
>>legend('Original signal','Recovered signal');
```

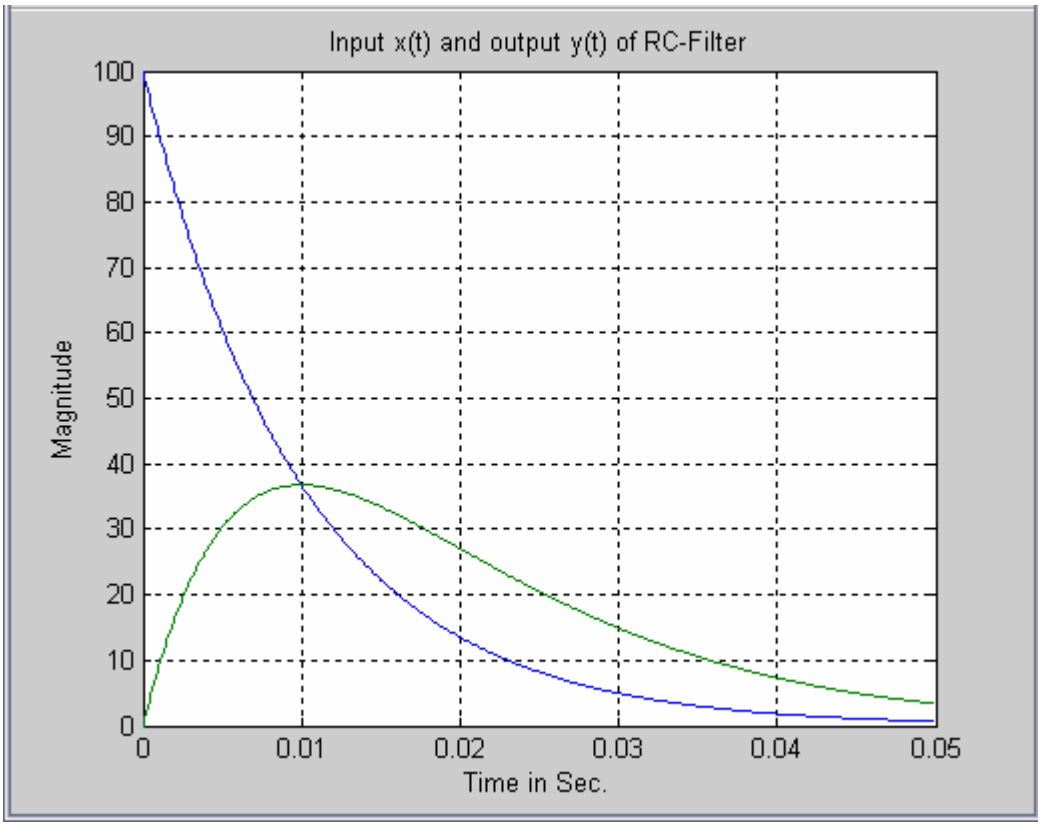


Práctica No. 11

- **Señal Filtro RC en Cascada**

Conjunto de comandos que permiten representar las señales en Tiempo Continuo propias de un filtro RC.

```
>>Fs=10000;
>>Ts=1/Fs;
>>N=500;
>>V=N*Ts;
>>R=100;
>>C=.0001;
>>a=1/(R*C);
>>h0=100;
>>x0=100;
>>t=0:Ts:V-Ts;
>>x=x0*exp(-a*t);
>>h=h0*exp(-a*t);
>>y=(x0*h0)*(t.*exp(-a*t));
>>plot(t,x,t,y)
>>grid
>>xlabel('Tiempo en Seg.')
>>ylabel('Magnitud')
>>title('Input x(t) and output y(t) of RC-Filter')
```



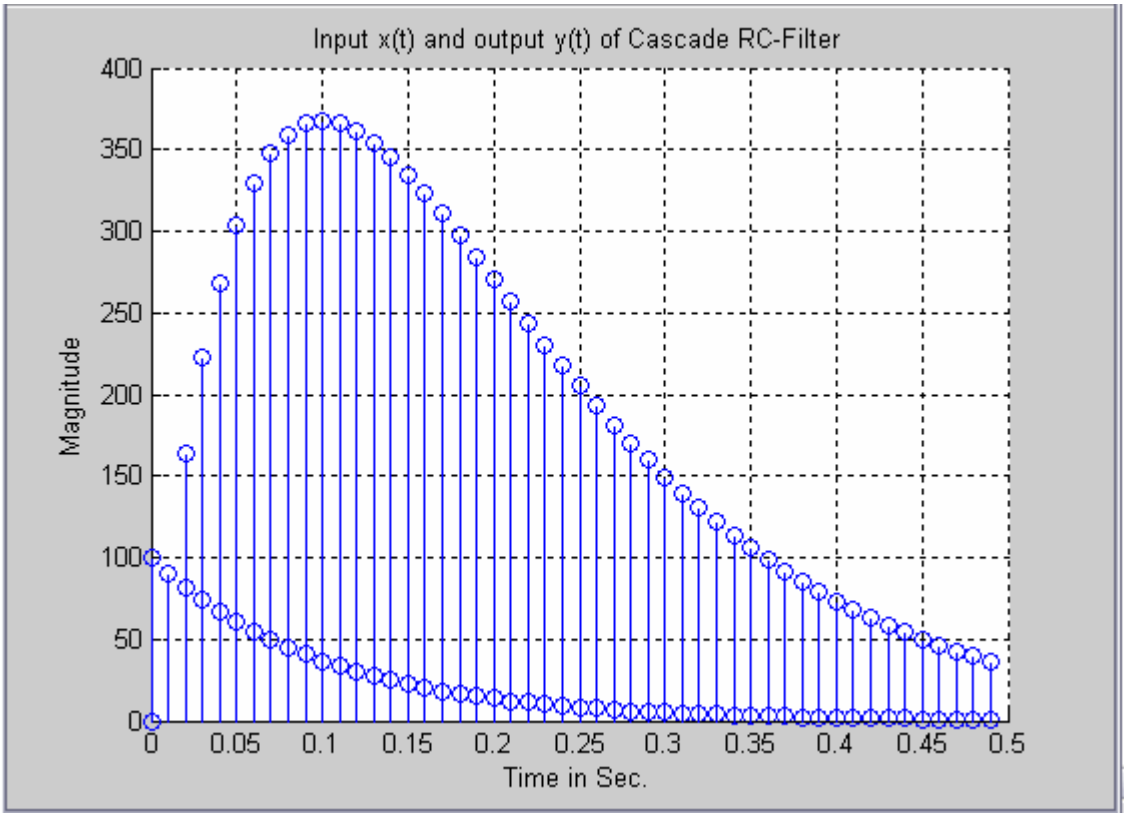
Práctica No. 12

- **Señal Filtro RC en Cascada**

Las salidas de respuesta de impulso en cada etapa de un RC-filtro de dos etapas de la cascada solían modelar un canal de UTP.

Conjunto de comandos que permiten representar las señales en Tiempo Discreto propias de un filtro RC.

```
>>Fs=100;
>>Ts=1/Fs;
>>N=50;
>>V=N*Ts;
>>R=10000;
>>C=10/1000000;
>>a=1/(R*C);
>>h0=100;
>>x0=100;
>>t=0:Ts:V-Ts;
>>x=x0*exp(-a*t);
>>h=h0*exp(-a*t);
>>y=(x0*h0)*(t.*exp(-a*t));
>>stem(t,x)
>>hold on
>>stem(t,y )
>>grid
>>xlabel('Time in Sec.')
>>ylabel('Magnitude')
>>title('Input x(t) and output y(t) of Cascade RC-Filter')
>>hold off
```


Input x(t) and output y(t) of Cascade RC-Filter
Magnitude
Time in Sec.
400
350
300
250
200
150
100
50
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5

Práctica No. 13

- **Controlando el Grupo de Delay (retrazo)**

Si rcosflt diseña el filtro automáticamente, entonces usted puede controlar el retraso de grupo del filtro, como descrito debajo. Si usted especifica su propio filtro de FIR, entonces rcosflt no necesita conocer su retraso de grupo.

El retraso de grupo del filtro es el tiempo entre la respuesta inicial del filtro y su respuesta máxima. El retraso de grupo de falta de la puesta en práctica es tres muestras de entrada. Para especificar un valor diferente, mídalo en períodos de símbolo de entrada y proporciónelo como el sexto argumento de entrada. Por ejemplo, el comando debajo especifica un retraso de grupo de seis muestras de entrada, que es equivalente a 6 *8 muestras de salida/1.

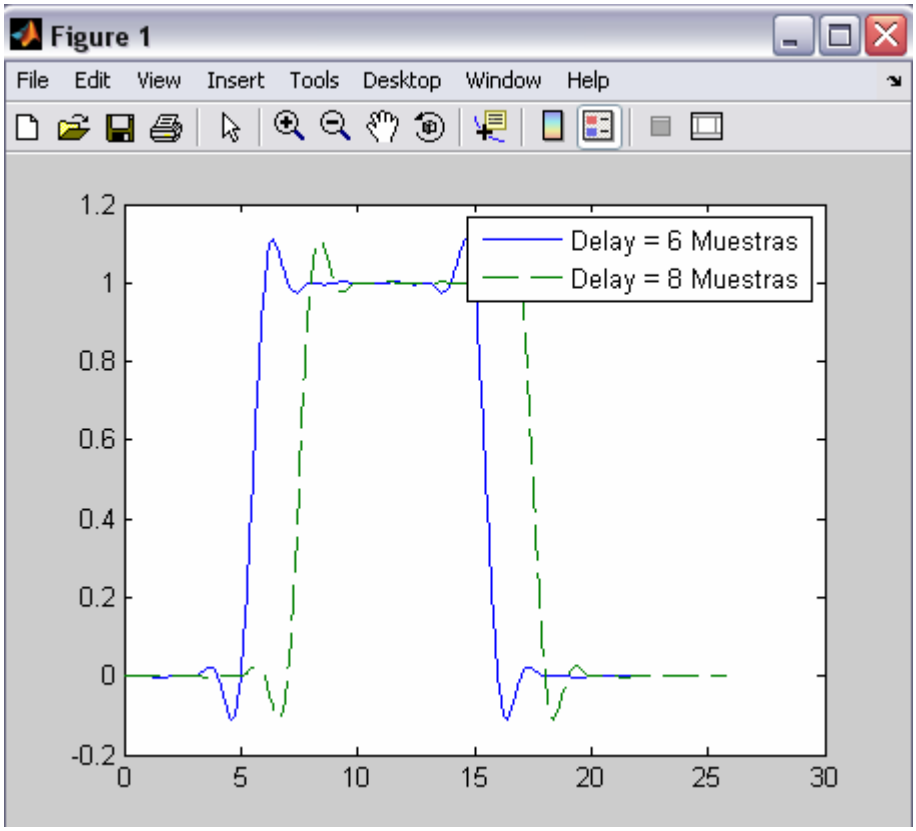
```
y = rcosflt([1;0;0],1,8,'fir',.2,6); % El Delay (retrazo) es equivalente a 6 muestras de entrada..
```

El retraso de grupo influye en el tamaño de la salida, así como la orden del filtro si rcosflt diseña el filtro automáticamente. Mirar la página de referencia para rcosflt para los detalles que se relacionan con la sintaxis que usted quiere usar.

Ejemplo:
Retrasos de Filtro de Coseno Aumentado

El código debajo filtra una señal que usa dos retrasos de grupo diferentes. Un retraso más grande causa un más pequeño error la respuesta frecuencias del filtro. El argumento muestra como las dos señales filtradas se diferencian, y el punto de salida indica que el primer pico ocurre varias veces para las dos señales filtradas. En el argumento, la línea sólida corresponde a un retraso de seis muestras, mientras la línea rota corresponde a un retraso de ocho muestras.

```
[y,t] = rcosflt(ones(10,1),1,8,'fir',.5,6); % Delay = 6 muestras
[y1,t1] = rcosflt(ones(10,1),1,8,'fir',.5,8); % Delay = 8 muestras
plot(t,y,t1,y1,'--') % Las dos curvas indican los diferentes Delays.
legend('Delay = 6 Muestras','Delay = 8 Muestras','Localizacion','NorthOutside');
peak = t(find(y == max(y))); % Tiempo donde la muestra el tope de la primera curva
peak1 = t1(find(y1 == max(y1))); % Tiempo donde la muestra el tope de la Segunda  curva
pt = [min(peak), min(peak1)] % Primer Tiempo máximo para ambas curvas
```



La salida será:

```
pt =

   14.6250   16.6250
```

Si Fs/Fd está al menos 4, entonces un valor de retraso de grupo de al menos 8 trabajos bien en muchos casos.

En los ejemplos de esta sección, Fs/Fd es 8.
Los retrasos de Seis Muestras (Línea Sólida) y Ocho Muestras (Línea Rota)

Observación: En esta práctica utilizamos un comando nuevo, para pasar una señal de entrada por un filtro de coseno.
Este comando es**: rcosflt**

Descripción:
La función rcosflt pasa una señal de entrada por un filtro de coseno levantado. Usted puede o dejar a rcosflt diseñar un filtro de coseno levantado automáticamente o usted puede especificar que el coseno levantado se filtra usando argumentos de entrada.

Práctica No. 14

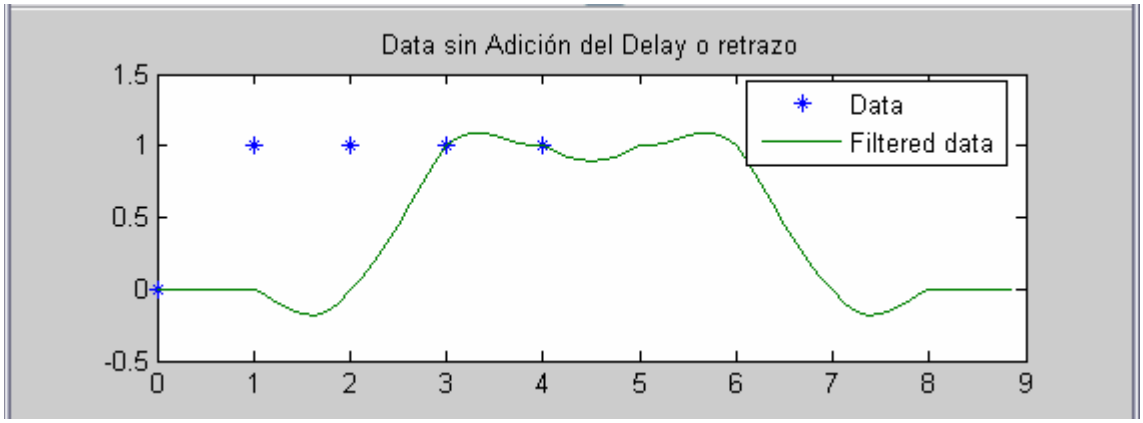
- **Compensación para grupos de Delays (retrazos) cuando Analizamos Data.**

La comparación de filtrado con datos infiltrados ósea no filtrados podría ser más fácil si usted retrasa la señal infiltrada por el retraso de grupo del filtro. Por ejemplo, supóngale usan el código debajo para filtrar **x** y producir **y**.

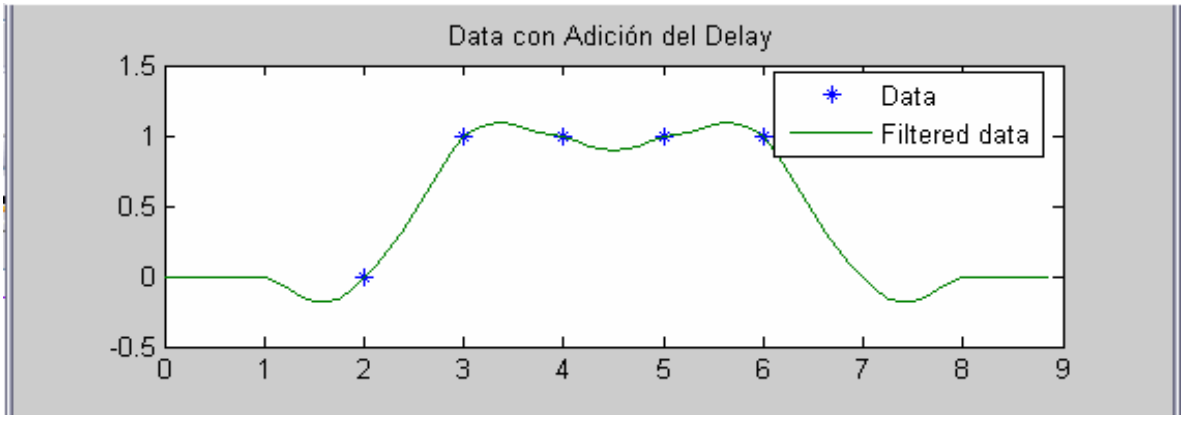
```
>>tx = 0:4; % Tiempo para la muestra de datos
>>x = [0 1 1 1 1]'; % Muestra de Datos Binarios
>>delay = 2; % El filtro de Data usa un retardo de 2 segundos,
>> [y,ty] = rcosflt(x,1,8,'fir',.3,delay);
```

Los elementos de **tx** y **ty** representan las veces de cada muestra de **x** y **y,** respectivamente. Sin embargo, y es retrasado en relación con **x**, entonces los elementos correspondientes de **x** y **y** no tienen los mismos valores de tiempo. La planificación y contra **ty** y **x** contra **tx** es menos útil que la planificación y contra **ty** y **x** contra una versión retrasada de **tx**.

```
>>% Top plot
>>subplot(2,1,1), plot(tx,x,'*',ty,y);
>>legend('Data','Filtered data');
>>title('Data sin Adición del Delay o retrazo');
```




```
>>% Bottom plot delays tx.
>>subplot(2,1,2), plot(tx+delay,x,'*',ty,y);
>>legend('Data','Filtered data');
>>title('Data con Adición del Delay');
```



Práctica No. 15

- **Ruido Blanco Gaussiano.**

La función de **wgn** genera matrices de manera arbitraria mediante la utilización de una distribución ruido blanco Gaussiana. Usted especifica el poder del ruido en el uno o el otro dBW (decibelios en relación con un vatio), dBm, o unidades lineales. Usted puede generar ruido verdadero o complejo.

Por ejemplo, el comando que se encuentra debajo genera una columna vector de longitud 50 conteniendo un verdadero ruido blanco Gaussiano cuyo poder es 2 dBW. La función asume que la impedancia de carga es 1 ohmio.

```
y1 = wgn(50,1,2);
```

Para generar un complejo ruido blanco Gaussiano cuyo poder es 2 vatios, a través de una carga de 60 ohmios, empleo cualquiera de los comandos que se encuentran debajo. El ordenamiento de las entradas no importa.

```
y2 = wgn(50,1,2,60,'complex','linear');
y3 = wgn(50,1,2,60,'linear','complex');
```

Para enviar una señal por un canal aditivo de ruido blanco Gaussiano, use la función de **awgn**.

awgn.Add white Gaussian noise to a signa (adiere ruido blanco gaussiano a la señal).

Sintaxis

- y = awgn(x,snr)
- y = awgn(x,snr,sigpower)
- y = awgn(x,snr,'measured')
- y = awgn(x,snr,sigpower,state)
- y = awgn(x,snr,'measured',state)
- y = awgn(...,powertype)

descripción

- y = awgn (x, snr)

Agrega que el ruido blanco Gaussiano al vector de señal x. El escalar snr especifica la proporción de señal-a-ruido por muestra, en dB. Si x es complejo, entonces awgn añade el ruido complejo. Esta sintaxis asume que el poder de x es 0 dBW.

- y = awgn (x, snr, sigpower)

Es lo mismo como la sintaxis encima, pero sigpower es el poder de x en dBW.

- y = awgn (x, snr, 'moderado')

Es lo mismo como y = awgn (x, snr), pero awgn mide el poder de x antes de la adición noise.

- y = awgn (x, snr, sigpower, el estado)

Es el mismo como y = awgn (x, snr, sigpower), pero awgn primero reinicializa el estado del generador de número aleatorio normal randn al estado de número entero.

- y = awgn (x, snr, 'moderado', estatal)

es lo mismo como y = awgn (x, snr, 'moderado'), pero awgn primero reinicializa el estado de generador de número aleatorio normal randn al estado de número entero.

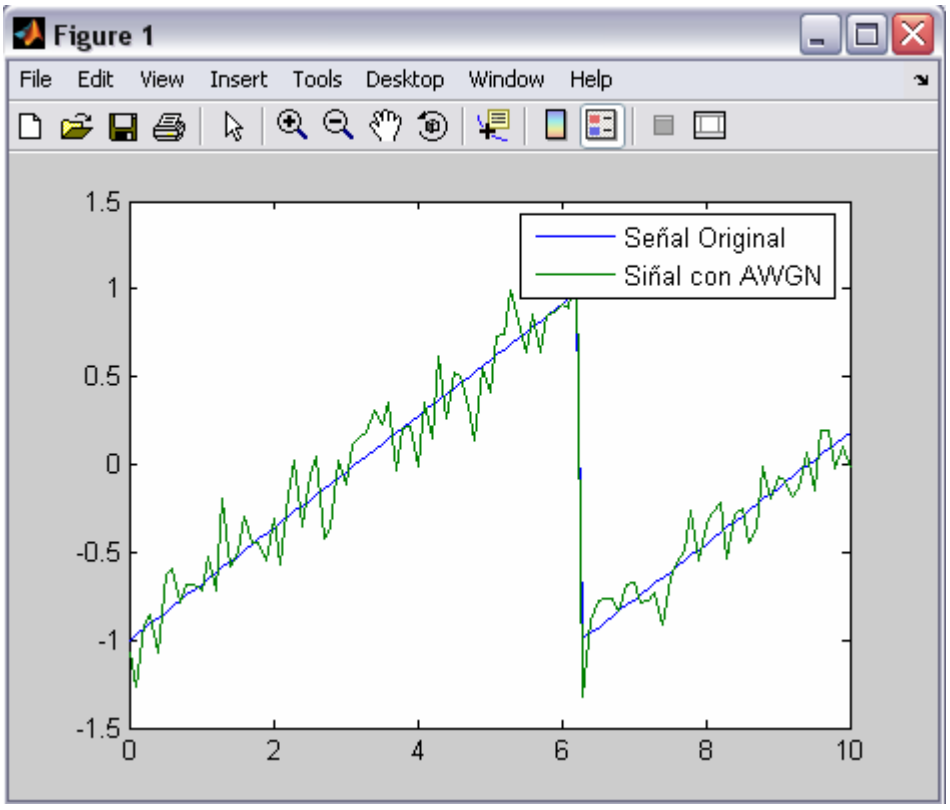
- y = awgn (..., powertype)

es lo mismo como la sintaxis anterior, pero la cuerda powertype especifica las unidades de snr y sigpower. Las opciones para powertype son 'db' 'y lineales'. Si powertype es 'db', entonces snr es medido en dB y sigpower es medido en dBW. Si powertype es 'lineal', entonces snr es medido como una proporción y sigpower es medido en vatios.

Ejemplo

Los siguientes comandos adieren un ruido blanco gaussiano a la señal tipo sierra. En la imagen se plotea la señal tipo sierra y la señal del ruido.

```
>>t = 0:.1:10;
>>x = sawtooth(t); % Creando señal tipo sierra.
>>y = awgn(x,10,'measured'); % Adiere el ruido blanco gaussiano a la señal.
>>plot(t,x,t,y) % Plotea ambas señales
>>legend('Señal Original','Siñal con AWGN');
```



Bibliografía.

1. **Instructivo de Matlab.**
2. **Señales y Sistemas**
   Autor: Alan V. Oppenheim, Alan S. Willsky
3. **Sistemas de Comunicaciones**
   Editora: MacGraw – Hill
4. **Comunications Systems**
   Autor: Simon Haykin, John Wiley & Sons