Capítulo 1. Introducción a Matlab
Aprendizaje y familiarización en el uso de Matlab

1. Objetivos

Este capítulo tiene como objetivo general la adquisición de los conocimientos básicos para el manejo de la herramienta matemática MATLAB.

Los objetivos concretos del capítulo son:

- El conocimiento del entorno de trabajo: descripción y localización de las herramientas más habituales.
- El conocimiento de los procedimientos de creación de vectores
- El aprendizaje de los modos de utilizar las operaciones básicas sobre las
- estructuras de datos
- El conocimiento de los procedimientos básicos de visualización
- El aprendizaje de los procedimientos para la creación de señales

2. Introducción

2.1. El programa Matlab

MATLAB es el nombre abreviado de “MATrix LABoratory”. MATLAB es un programa para realizar cálculos numéricos con vectores y matrices. Como caso particular puede también trabajar con números escalares (reales y complejos), con cadenas de caracteres y con otras estructuras de información más complejas. Una de las capacidades más atractivas es la de realizar una amplia variedad de gráficos en dos y tres dimensiones. MATLAB tiene también un lenguaje de programación propio. Este apartado hace referencia a la versión 7.0 de este programa (también llamada release 14), aparecida a mediados de 2004.

MATLAB® es, básicamente, un lenguaje de altas prestaciones para la computación en todas aquellas áreas basadas en procesamiento de datos, sean de la índole que sean. Además del lenguaje, MATLAB integra en un entorno de uso bastante sencillo y amigable, una gran cantidad de capacidades de cómputo,visualización y programación. Los usos más típicos de MATLAB son:

- Análisis Matemático y Simulaciones Numéricas.
- Cálculo Simbólico.
- Desarrollo y Test de Algoritmos.
- Modelado de Sistemas.
- Análisis Estadísticos y Modelos Predictivos.
- Gráficos Científicos e Ingenieriles.
- Desarrollo de Aplicaciones y Productos Finales incluyendo GUI
- (Graphical User Interfaces)
- etc.

MATLAB es en última instancia un entorno de desarrollo interactivo cuyo elemento básico es un array. Esto es, MATLAB entiende los vectores y las matrices de la misma forma que C o Fortran entienden las variables. Sin la necesidad de desarrollar programación basada en bucles anidados para realizar operaciones entre arrays.

Desde su aparición, MATLAB ha ido renovándose y creciendo gracias a las contribuciones de sus usuarios. Producto del interés que distintas áreas científico−técnicas en MATLAB han surgido gran cantidad de funciones específicas para cada área que vienen agrupadas en un paquete denominado *toolbox*. Estas *toolboxes* consisten en colecciones de funciones para MATLAB *m−functions* desarrolladas para resolver problemas de un tipo particular. De entre esas área podemos citar:

- Procesamiento de Señal.
- Sistemas de Control.
- Redes Neuronales.
- Procesamiento de Imágenes.
- Lógica Difusa.
- Caracterización y Modelado de Sistemas.
- Economía.
- Ecuaciones Diferenciales Multidimensionales.

El sistema completo MATLAB consiste de 5 bloques diferenciados y que pueden funcionar independientemente.

• **Entorno de Desarrollo**: Es el conjunto de herramientas que le permite usar los archivos y funciones de MATLAB. La mayoría de estas herramientas son interfaces gráficas para el usuario. En concreto la de más alto nivel es el denominado MATLAB desktop, que incluye al Command Window, al Command History, al Workspace viewer y al Path Browser.
• **Librería de Funciones**: Esta librería es una amplísima colección de algoritmos de cómputo de muy diversa complejidad, desde sumas, restas y funciones trigonométricas hasta obtención de FFTs, etc.
• **Lenguaje de MATLAB**: Es un lenguaje de programación de alto nivel cuyo elemento básico es un array. Como buen lenguaje de programación incluye sentencias de control de flujo, estructuras, y objetos.
• **Sistema de Gráficos**: Que le permite visualizar sus datos en una gran cantidad de representaciones distintas. Gracias a comandos de alto nivel Ud. podrá visualizar gráficos bi-dimensionales y tri-dimensionales, imágenes, animaciones, etc. Además incluye comandos de bajo nivel que le permitirán personalizar sus representaciones gráficas y construir interfaces gráficas para usuarios.
• **API (Application Program Interface**): Esta es una librería que le permitirá escribir sus funciones en C o Fortran y hacerlas interactuar con códigos escritos en el lenguaje nativo de MATLAB.

2.2. Entorno de Trabajo

Al arrancar MATLAB se abre una ventana similar a la mostrada en la Figura 1. Ésta es la vista que se obtiene eligiendo la opción Desktop Layout/Default, en el menú View.

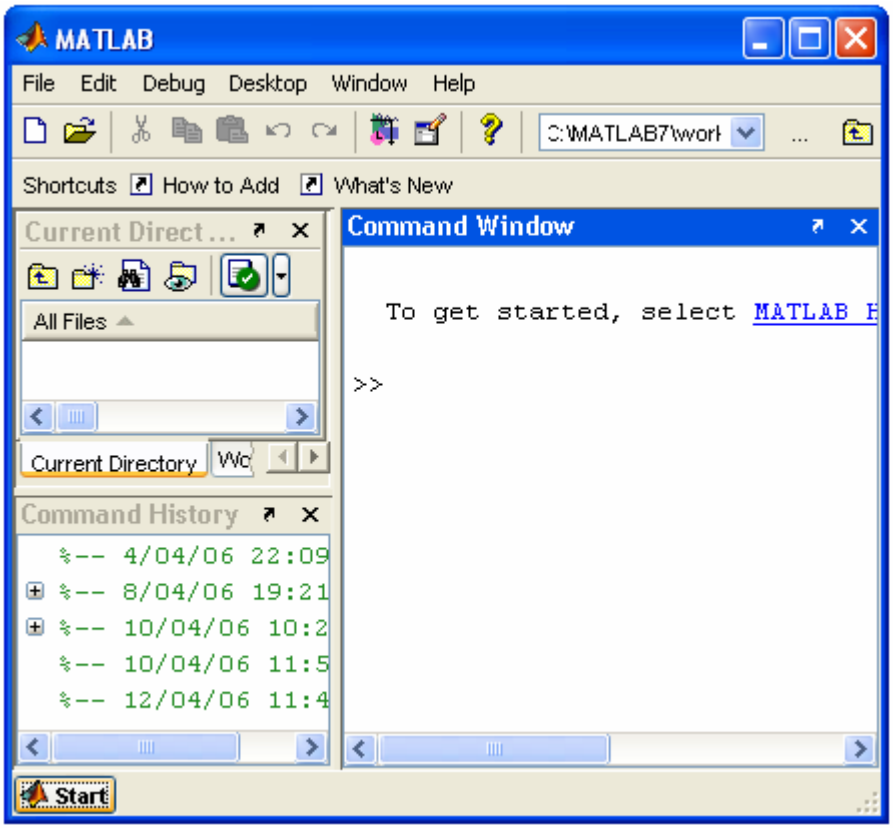


Figura 1.1. Ventana principal 7.0

La parte más importante de la ventana inicial es Command Window, en esta subventana es donde se ejecutan los comandos de MATLAB (>> indica que el programa está preparado para recibir instrucciones).

En la parte superior izquierda de la pantalla aparecen dos ventanas que se pueden alternar: Current Directory y Workspace. La ventana Current Directory muestra cúal es el directorio de trabajo actual (haciendo doble clic sobre alguno de los ficheros *.m del directorio actual se abre el editor de ficheros de MATLAB). La ventana Workspace contiene información sobre las variables que se hayan definido en la sesión.

En la parte inferior derecha aparece la ventana Command History que muestra los últimos comandos ejecutados en la ventana Command Window.

En la parte inferior izquierda de la pantalla aparece el botón Start. Stara da acceso inmediato a ciertas capacidades del programa. La Figura 2 muestra las posibilidades de Start/MATLAB.

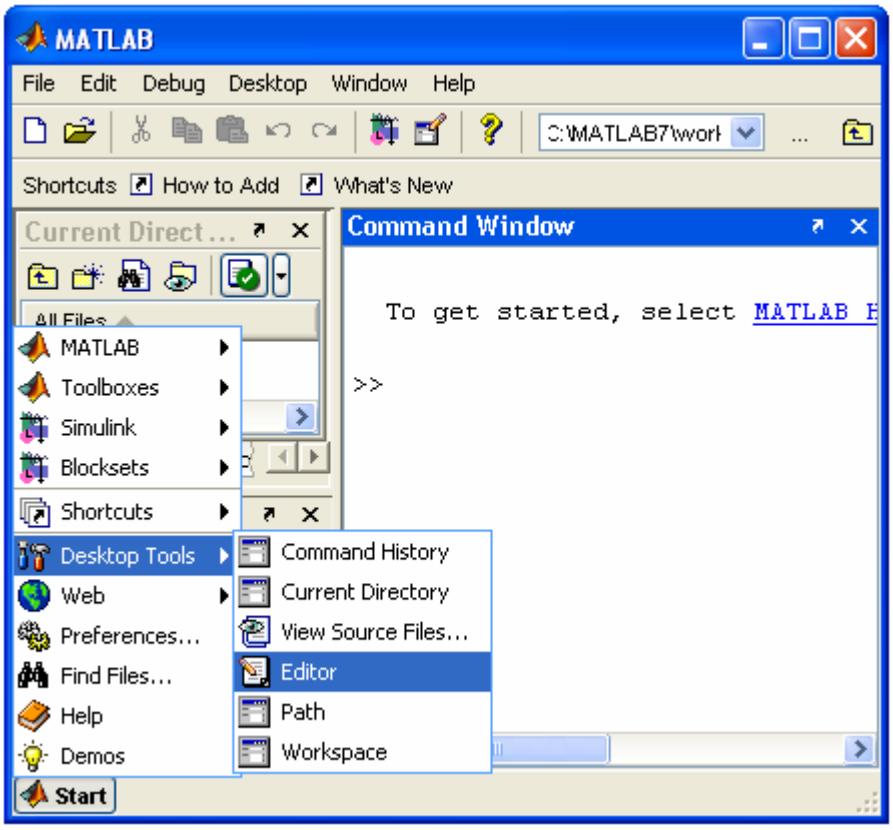


Figura 1.2. Menú Start

3. Las Bases del Lenguaje

3.1 Introducción

a) Variables

MATLAB no requiere ninguna declaración previa a asignación para crear variables. Cada vez que encuentra una nueva declaración, crea una variable la guarda en memoria. Además si una variable ya existe pero se reasigna y esa nueva operación cambia el tamaño que esa variable ocupaba en memoria, MATLAB redimensiona también el espacio de memoria para guardar tal variable. Los nombres de las variables deben empezar por una letra, seguido de cualquier combinación de letras, números y underscores. MATLAB solo lee los primeros 31 caracteres en el nombre de una variable. Además MATLAB es sensible a mayúsculas y minúsculas.

```
>> a = 5
>> A = 6  % Observando la ventana Workspace, se
crean dos variables distintas.
>> a = 7  % Resasigna el valor de la variable A
```

b) Números

MATLAB usa la notación decimal convencional, con opciones para la ubicación del punto decimal. La notación científica emplea la letra **e** para las potencias de 10 mientras que para números imaginarios se pueden usar indistintamente i o j. Internamente todos los números se almacenan según el formato *long* definido por el standard de la IEEE para notación en punto flotante.
Esto es una precisión de los 16 decimales más significativos con valores comprendidos entre $\left[10^{-308}, 10^{308}\right]$.

c) Operadores

Las operaciones en MATLAB usan la notación convencional para los operadores aritméticos y las reglas de precedencia.

- + Suma.
- - Resta.
- * Producto.
- / División.
- ^ Exponenciación.
- ' Transposición.
- ( ) Especificación del orden de las operaciones.

d) Funciones

MATLAB incorpora un gran número de funciones matemáticas básicas, como raíz cuadrada **sqr( )**, valor absoluto **abs( )**, exponencial **exp( )**, seno **sin( )**, coseno **cos( )**, tangente **tan( )**, etc.

```
>> help elfun          % Lista la funciones mas elementales
>> help real           % Información acerca de la función real
```

MATLAB también incorpora funciones especiales, para ello tecleamos lo siguiente:

```
>> help specfun       % Lista las funciones especiales
>> help besselj       % Información acerca de la función besselj
```

Por último, MATLAB también incorpora funciones para matrices, para obtener información sobre esto, realizamos lo siguiente:

```
>> help elmat         % Lista las funciones para matrices
>> help zeros         % Información acerca de la función zeros
```

Existen funciones especiales definidas para poder substituir en nuestras expresiones aquellas constantes que usamos normalmente por su valor “textual” en lugar del valor numérico. De entre ellas destacamos:

- **pi** 3.14159265…
- **i**
- **j** lo mismo que **i**
- **Inf** infinito

• **NaN** No es un número, para operaciones del tipo 0/0.
• **ans** (de *answer*) Es una variable dinámica que MATLAB asigna al valor de última operación efectuada.
• **eps** la mínima distancia entre números que puede definirse
• **realmax**. El mayor número que podemos usar ~1.797610308
• **realmin** El menor número que podemos usar ~2.225110-308

3.2. Creación de Variables

a) Escalares

Como dijimos en la introducción MATLAB es un lenguaje de programación especialmente orientado al cálculo matricial. Tanto que una variable escalar, sea del tipo que sea, es considerada por MATLAB como un array de tamaño [1x1] Por tanto, las operaciones entre matrices o vectores se definen de la misma manera que las operaciones entre escalares.

A diferencia de muchos lenguajes de programación, en MATLAB, no necesita declarar las variables a usar en la cabecera de un fichero antes de asignarlas, sino que creación de la variable en memoria, asignación del tipo de dato a almacenar y valor del dato se hacen todo en uno. Simplemente asignamos la parte a la derecha de un signo igual, al nombre de variable declarado a la izquierda. Realizamos lo siguiente en la vemtana Command Windows de MATLAB:

```
>> a=5     %Observamos la ventana workspace
a =
     5

>> whos    %Vemos las variables que hay en memoria
  Name      Size                    Bytes  Class
  a         1x1                         8  double array

Grand total is 1 element using 8 bytes

>> clear a  %Eliminamos la variable de memoria
```

b) Vectores

La creación de un escalar, un vector o una matriz, se hacen mediante procedimientos idénticos. Simplemente escribimos el nombre de la variable a la izquierda de la expresión y a lo que lo queremos asignar, a la derecha. Realizamos los siguientes ejemplos

```
>> v=[1 3 5 7]
v =
1     3     5     7
>> v(5)=9  %Aumentamos la dimensionalidad del vector
añadiendo índices
v =
1     3     5     7     9
>> a=[9 10]
a =
9    10
>> b =[v a]
b =
1     3     5     7     9     9    10
>> length(b) %length visualiza la longitud de un vector
```

Muchas veces interesa definir vectores en los que sus elementos están equiespaciados. Un ejemplo típico es cuando queremos generar la variable tiempo t para análisis de sistemas dinámicos. MATLAB nos evita tener que generar a mano todos los puntos de este eje de tiempos mediante el constructor de vectores "**:**". La sintaxis de este constructor es

```
>> variable= valor_inicial : paso : valor_final ;
```

Por ejemplo, el siguiente código, genera un eje de tiempos formado por valores tomados cada 1s. Empezando por 0 y terminando por 10.

```
>> Tstart = 0
>> Tend = 10
>> Tstep = 1    %Si no se especifica el paso se tomará 1
>> t1 = Tstart:Tstep:Tend
>> t2 = Tstart:Tend   %t1 = t2 ya que el paso es 1
```

c) Matrices

Las matrices se definen introduciendo los elementos fila a fila. Las filas se especifican como los vectores y la separación entre filas se hace mediante ";"

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

MATLAB además posee constructores especiales:

```
>>M = [ ];                    % Matriz vacía
>>M = zeros(n,m);             % Matriz de n x m ceros
>>M = ones(n,m);              % Matriz de n x m unos
>>M = eye(n);                 % Matriz identidad de rango n
```

3.3. Guardar Variables y Estados de una Sesión.

Para poder guardar el estado de una sesión de trabajo existe el comando `save`. Si teclea este comando antes de abandonar el programa, se crea en el directorio actual un fichero binario llamado `matlab.mat` (o `matlab`) con el estado de la sesión. Dicho estado se recupera con el comando `load`.

Estos dos comandos también pueden ser utilizados para guardar estructura de datos, de tal forma:

```
>> save nombrefichero A x y    %guarda en nombrefichero.mat (o
nombrefichero)
                                %las variables A, x e y
>> load nombrefichero          %Recupera en otra sesión las
variables A,x e y
```

3.4. Manipulación de Matrices

Realizamos los siguientes ejemplos para ver como tratar las matrices en MATLAB:

```
>> A = [1 2];
>> B = [3 4];
>> C = [A ; B]          %CONCATENACION
C =
     1     2
     3     4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>> size (C)  %OPERADOR SIZE; devuelve un vector fila
             %de dos componentes, numero de filas y de columnas
ans =
     2     2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>> A=[1 2 3 4 5];
>> C=[A; A; A; A; A];
>> sum(C)    %COMANDO SUM genera un vector fila donde cada
             %elemento es la suma de todos los elementos en su
             %columna en la matriz que se le pasa como argumento

ans =
5 10 15 20 25
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
>> A =[1 2;3 4]
>> B = A'               %B es la traspuesta de A
>> C = transpose(A)     %Equivalente a la línea anterior
>> det (A)              %calcula el determinante de A
>> D = inv(A)           %D es la matriz inversa de A
```

a) Operaciones con Matrices

MATLAB puede operar con matrices por medio de operadores y por medio de funciones. Los operadores matriciales de MATLAB son los siguientes:

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

Para ver como actúan estos operadores tecleamos lo siguiente:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SUMA, RESTA, MULTIPLICACIÓN Y DIVISION DE UNA MATRIZ CON UN
% ESCALAR
% se realiza elemento a elemento
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

>> A = [2 4; 6 8];

>> B = A + 2
B =
     4     6
     8    10
>> C = A – 2
C =
     0     2
     4     6
>> D = A * 2
D =
     4     8
    12    16
>> E = A / 2
E =
     1     2
     3     4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SUMA, RESTA, MULTIPLICACIÓN Y DIVISION DE 2 MATRICES
% los operadores + - * ^ y / tienen que cumplir dimensiones
% propias del algebra de matrices
% los operadores .* y ./ hacen la multiplicación y división
% elemento a elemento.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

>> [1 2 3 4]^2
??? Error using ==> ^
Matrix must be square.
>> [1 2 3 4].^2
ans =
1 4 9 16
>> [1 2 3 4]*[1 -1 1 -1]
??? Error using ==> *
Inner matrix dimensions must agree.
>> [1 2 3 4].*[1 -1 1 -1]
ans =
1 -2 3 -4
```

3.5. Definición de funciones con M-files

Los ficheros M son macros de comandos de MATLAB almacenadas como ficheros de texto con extensión ".m" o sea nombre_fichero.m. Un M-file puede ser una función con variables de entrada y salida o simplemente una lista de comandos (script de comandos de MATLAB). Para editar una función escribiremos en la ventana de comandos el comando edit, a continuación.

La sintaxis para la definición de funciones en MATLAB con M-files es la siguiente:

```
>> function [variables_salida] = nombre_funcion
(parametros_entrada)
```

Para editar una función escribiremos en la ventana de comandos el comando edit, a continuación escribimos la función, la guardamos como nombre_funcion.m y por último para invocar la función desde la ventana de comandos tecleamos la siguiente línea:

```
>>[variables_salida] = nombre_funcion (parametros_entrada)
```

Ejercicio 1.1

Cree un fichero en su directorio de trabajo de nombre comandos.m que devuelva la suma, resta, multiplicación y división de dos números introducidos como argumentos.

```
function [suma, resta, multiplicacion, division]=comandos(var_a,
var_b)
%COMANDOS retorna la suma, resta, multiplicación y división de dos
variables.

suma=var_a+var_b;
resta=var_a-var_b;
multiplicacion=var_a*var_b;
division=var_a/var_b;
```

3.6. Representación

MATLAB contiene un amplio set de herramientas para la visualización de vectores y matrices, asi para incluir anotaciones en ella. Mostramos las más importantes:

- **`plot (x,y)`** : Dibuja el vector x frente al vector y; Para este comando existen una gran cantidad de modificadores y parámetros, para verlos (>> help plot).
- **`xlabel ( ), ylabel( ), zlabel ()`** para poner texto a los ejes
- **`title( ) :`** Pone título a la gráfica

Ejercicio 1.2.

Cree un fichero en su directorio de trabajo llamado `ejercicio1_2.m`, este fichero contendrá una secuencia de comandos que haga lo siguiente:

Genere un vector de tiempos uniforme que parta de $t = 0s$ y llegue a $t = 1s$ y que contenga 1000 muestras. Construya una función seno y_1 en esa base de tiempo de amplitud 1 y frecuencia 5Hz. Construya una función y_2 exponencial decreciente sobre esa base de tiempos con una constante de tiempo τ igual a 100ms. Genere una tercera forma de onda y_3 como producto de las dos anteriores .Use plot para dibujar las formas de onda y_1, y_2 e y_3 en un único dibujo, añada un título a la gráfica y etiquete los ejes x e y.

ejercicio1_2.m

```
Tinicial = 0;
Tfinal = 1;
Npuntos = 1000;
paso = (Tfinal-Tinicial)/Npuntos;
t = Tinicial:paso:Tfinal-paso;
y1 = 1*sin(5*2*pi*t);
tau = 200e-3;
y2 = exp(-t/tau);
y3 = y2 .* y1;
plot(t,y1,t,y2,t,y3)
xlabel('Tiempo')
ylabel('Tension')
title('Ejercicio1.2')
```

```
>> ejercicio1 2
```

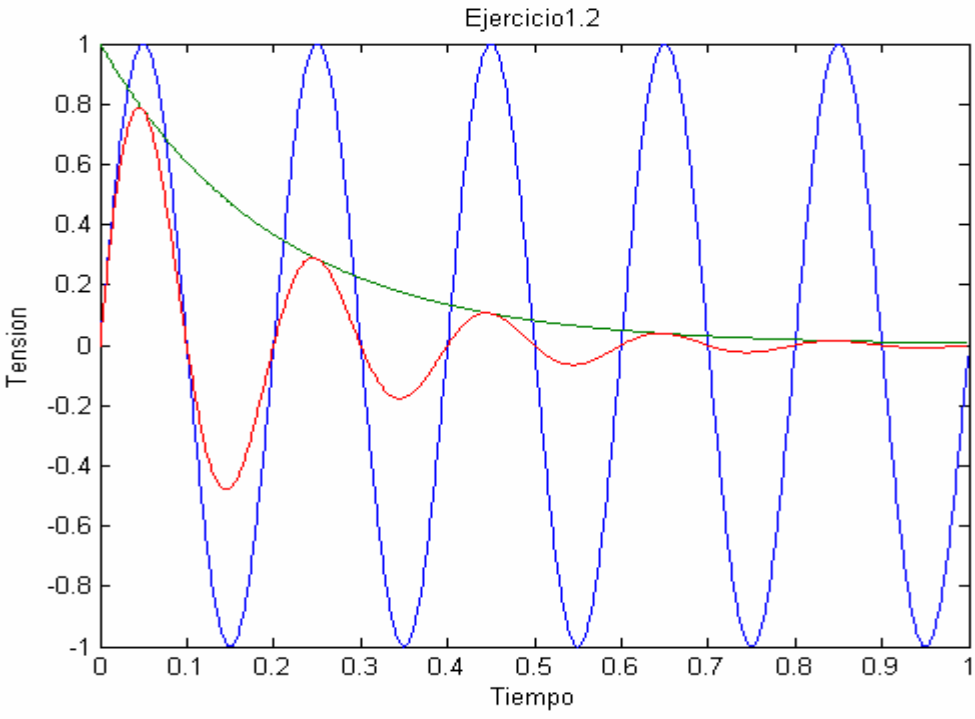


Figura 1.3. Resultado Ejercicio 1.2

Para editar propiedades de la gráfica, añadir comentarios y flechas de marcación o cambiar los colores use el menú `Tools => Edit plot` y a continuación haga doble clic con el ratón sobre la gráfica.

Para dibujar una función de dos variables primero debemos generar los vectores correspondientes a esas variables. Para definir grids MATLAB incorpora el comando `meshgrid(x,y)`. Este comando transforma los dominios definidos por los vectores x e y en un grid de puntos. Así, si hacemos `[X,Y]=meshgrid( x,y)`, producimos una matriz X cuyas filas son copias del vector x y una matriz Y cuyas columnas son copias del vector y.

Dos son los comandos más básicos para el dibujo de superficies en MATLAB, los comandos `mesh()` y `surf()`. El primero de ellos, `mesh( )`, produce superficies definidas por líneas coloreadas de acuerdo a la altura z de los puntos que conectan y al mapa de colores definido v, mientras que `surf( )` rellena con colores.

3.7. Programación y Control

MATLAB, como cualquier lenguaje de programación, contiene, además de su amplía gama de funciones internas, una serie de constructores que le permiten controlar el flujo de ejecución de sus programas. En este apartado veremos los más destacados.

a) IF

La sentencia if evalúa una expresión lógica y ejecuta una serie de líneas (sentencias) si esa expresión es verdadera.

Opcionalmente puede usarse elseif y else para la ejecución de grupo de órdenes alternativas. MATLAB usa la palabra clave end al final del bloque a ejecutar para cerrarlo (a diferencia de otros lenguajes en los que se emplea {}) .

```
if condicion
   sentencias
end
```

```
if condicion1
bloque1
elseif condicion2
bloque2
elseif condicion3
bloque3
else % opción por defecto para cuando no se
cumplan las condiciones 1,2,3
bloque4
end
```

Debemos tener cuidado con los argumentos dentro de `if` si la condición se aplica sobre matrices, se hace elemento a elemento. Así `if A==B` no devuelve un `1` o un `0` sino una matriz de unos y ceros correspondiente al chequeo de la condición elemento a elemento. Además si A y B son de distinto tamaño, el programa devuelve un error. Si queremos evaluar si dos variables son exactamente iguales debemos emplear siempre la función `isequal( )` que toma dos argumentos y devuelve un `1` en caso de igual. Así para dos matrices podríamos escribir

```
>> if isequal (A,B)
```

El conjunto de operadores lógicos y relacionales disponibles son:

- **Relacionales: <, >, >=, <=, ==, ~=.** Correspondientes respectivamente a menor, mayor, mayor o igual, menor o igual, igual y distinto.
- **Lógicos: &, |, ~, xor**. Correspondientes a AND, OR, NOT, y XOR respectivamente.

b) SWITCH Y CASE

Las sentencias `switch` y case ejecutan una serie de ordenes u otras dependiendo del resultado de evaluar una expresión. El final de un bloque `switch` se especifica mediante la palabra reservada `end`. Al contrario de lo que ocurre en C, en MATLAB la sentencia `break` no es necesaria para salir del bloque una vez se ha detectado una condición ya que MATLAB solo ejecutará la primera de las coincidencias.

```
switch switch_expresion
case case_expr1,
bloque1
case {case_expr2, case_expr3, case_expr4,...}
bloque2
...
otherwise, % opción por defecto
bloque3
end
```

c) FOR

Cuando queremos que algo se ejecute un número determinado de veces lo más cómodo es incluirlo en un bucle `for`, que debe ser terminada con un `end`. Estos bucles no precisan la orden de autoincremento o decremento como los que escribimos en lenguajes basados en C, sino simplemente los valores iniciales y finales.

```
for i=valor_incial:paso:valor_final
sentencias
end
```

d) WHILE

Un bucle tipo `while` repetirá una serie de ordenes indefinidamente hasta que se cumple una condición lógica que se chequea al empezar a ejecutarse una nueva iteración.

```
while condicion
sentencias
end
```

e) BREAK

Al igual que en C/C++/Java, la sentencia `break` hace que se termine la ejecución del `for` y/o `while` más interno de los que comprenden a dicha sentencia.

4.Representación de una Señal Analógica.

4.1. Entorno de Simulación

Si procuramos modelar una señal analógica en un ordenador digital, el entorno de simulación impone ciertas limitaciones en el grado de precisión que puede ser alcanzado al representar la señal analógica. Este apartado describe las características de las señales analógicas simuladas en un ordenador digital y trata su significado desde la perspectiva del usuario.

En MATLAB, una función f(t) es definida como una cadena de puntos, los cuales representan la amplitud de la función en los instantes de muestreo. Por ejemplo la cadena $[f(t_1), f(t_2),..., f(t_n)]$ representa $f(t)$ en los instantes de muestreo $t_1, t_2, ..., t_n$. Dependiendo del número de puntos de la cadena de datos, el resultado se corresponderá con una versión muestreada distinta de la función. Para realizar esta observación, tecleamos el siguiente comando en MATLAB:

```
>> x = sin(t)
?? undefined function or variable 't'.
```

Matlab no tiene un entorno de simulación simbólico como MAPLE. Como resultado, el comando anterior generará un mensaje de error, ya que el argumento del seno, no ha sido previamente definido. Para definirlo, solo hay que especificar los instantes de muestreo de t. Por ejemplo, para dibujar una senoide de 50 Hz:

```
>> fs = 100;          % Frecuencia de muestreo ( 2*F_max = 2*50 )
>> Tinicial = 0;
>> Ts = 1/fs;         % Tiempo de muestreo
>> Tfinal = 0.1;
>> t = [Tinicial:Ts:Tfinal]
>> x = sin(2*pi*50*t)
>> plot (t,x);
```

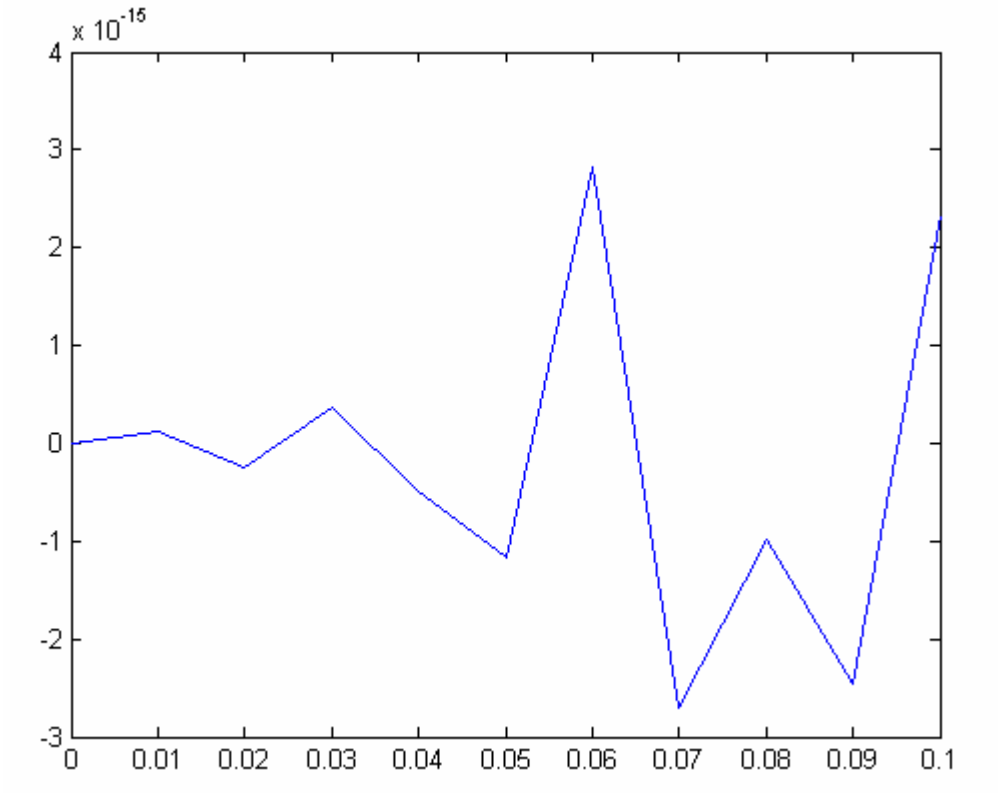


Figura 1.4. Señal senosoidal de 50 Hz muestrada a Fs = 100Hz

Por lo tanto, ¿cómo se puede generar una forma de onda analógica? Una respuesta simple es que no se puede generar una verdadera señal analógica. Consecuentemente, solamente podremos hablar de señales "pseudo-analógicas", que han sido muestreadas con una tasa suficientemente alta, con lo cual la señal muestreada representará exactamente la forma de onda analógica.

MATLAB wave_gen genera forma de ondas representando líneas de código binario exactamente de esta manera. Por ejemplo, si especificamos una tasa binaria de 50 Hz, la frecuencia de muestreo se cambia a 500 Hz dependiendo del experimento. La regla sería la siguiente $f_s = Kf_i$ donde f_i es la frecuencia de la señal de entrada o la tasa binaria de datos y $K \geq 10$ es la constante de muestreo; es el compromiso entre el tamaño de la cadena de datos y la precisión.

Para generar la señal senosoidal de antes de 50 Hz, ejecutamos el siguiente código:

```
>> fs = 500;          % Frecuencia de muestreo (10*Fmax =10*50 )
>> Tinicial = 0;
>> Ts = 1/fs;         % Tiempo de muestreo
>> Tfinal = 0.1;
>> t = [Tinicial:Ts:(Tfinal-Ts)]
>> x = sin(2*pi*50*t)
>> plot (t,x);
```

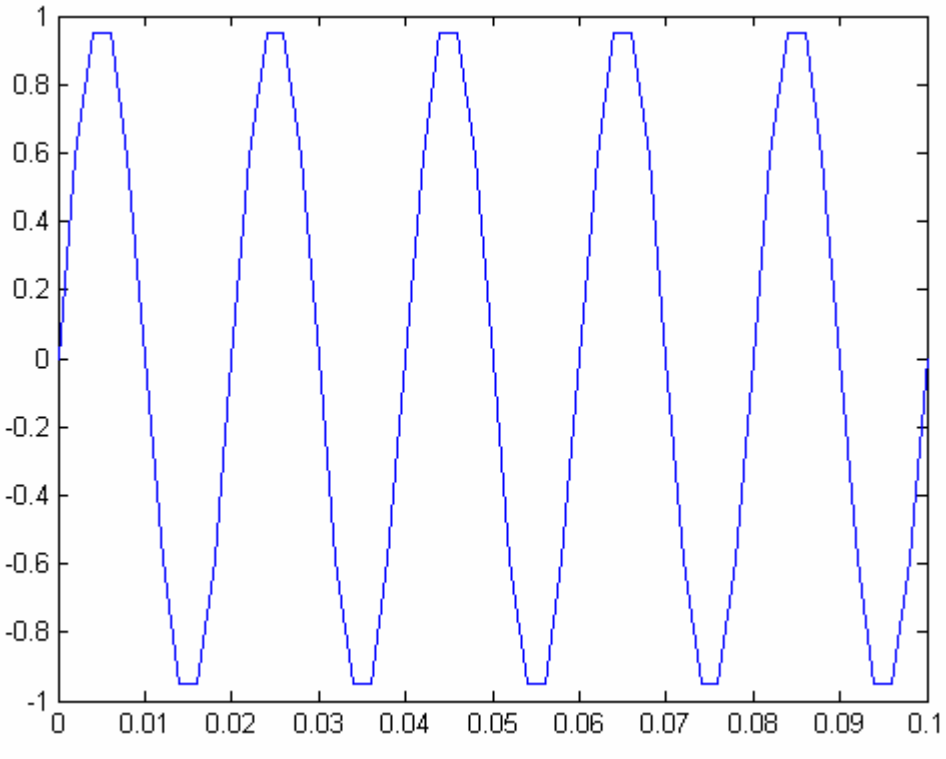


Figura 1.5. Señal senosoidal de 50 Hz muestrada a Fs = 500Hz

```
>> fs =1000;          % Frecuencia de muestreo ( 20*Fmax = 20*50 )
>> Tinicial = 0;
>> Ts = 1/fs;         % Tiempo de muestreo
>> Tfinal = 0.1;
>> t = [Tinicial:Ts:(Tfinal-Ts)]
>> x = sin(2*pi*50*t)
>> plot (t,x);
```

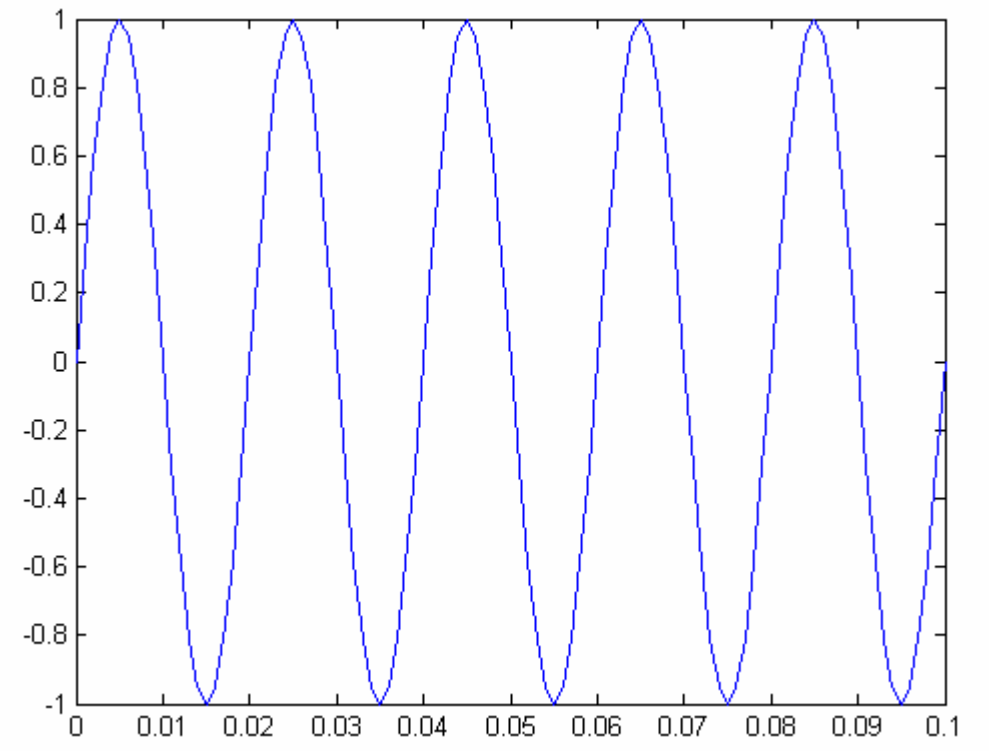


Figura 1.6. Señal senosoidal de 50 Hz muestrada a Fs = 1000Hz

Ejercicio Final.

Enunciado

En este apartado, se creará una función en MATLAB ***generador*** *que simulará un generador de señales analógicas. La función* ***generador*** *toma los siguientes argumentos:*

```
% Uso:
%  [t,y]  = generador (f, Fs, tipo, minval, maxval, Tinicial, Tfinal)
% donde :
%          f        =    Frecuencia (Hz)
%          Fs       =    Frecuencia de Muestreo (Hz)  (Nos da la resolución)
%          tipo     =    Tipo de onda a generar entre las siguientes:
%                        'seno','coseno','triangulo','cuadrado'
%          minval   =    Mínimo valor de la función en el eje y
%          maxval   =    Máximo valor de la función en el eje x
%          Tinicial  =    Instante de tiempo en el que comienza la función
%          Tfinal    =    Instante de tiempo en el que termina la función
```

A partir de esta función genere las siguientes señales:

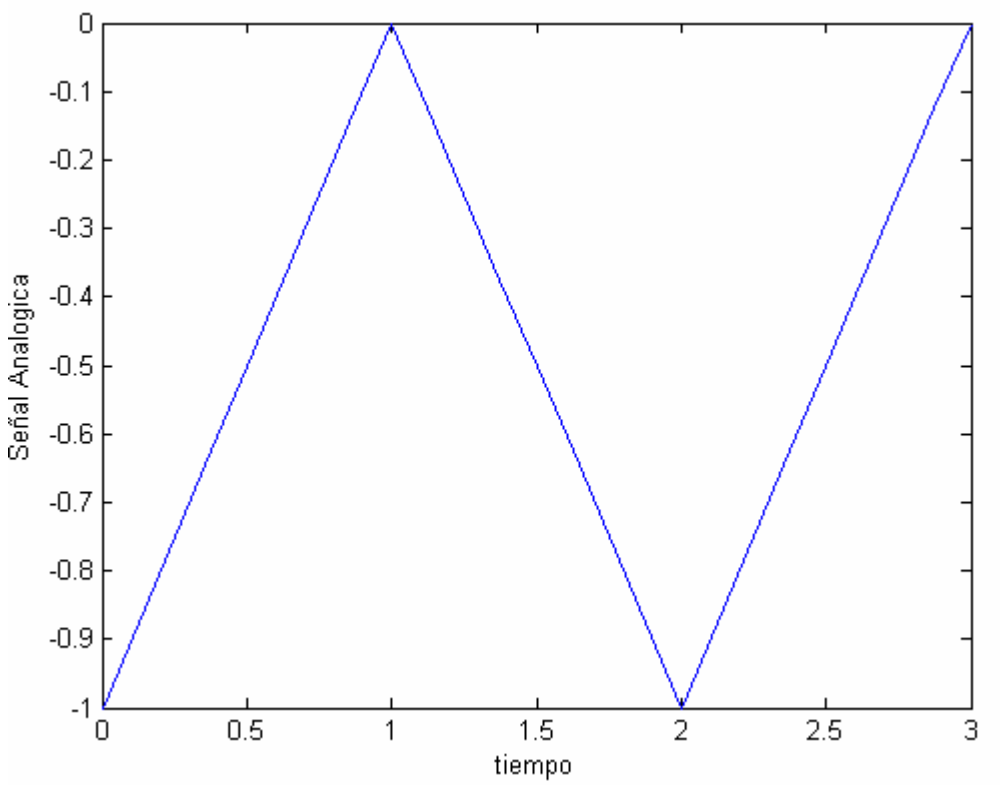
a) Señal triangular con f=0.5 Hz , minval=-1, maxval=0;
b) Seno de f=50 Khz y de amplitud unidad

Resultado

```
function [t, x] = generador(f, Fs, wf, minval, maxval, Tinicial,
Tfinal)
t = [Tinicial: 1/Fs: (Tfinal-(1/Fs))];
switch wf
    case 'seno'
        x = 0.5*(maxval-minval) * sin(2*pi*f*t) + .5*(maxval-
minval)+minval;
    case 'coseno'
        x = .5*(maxval-minval) * cos(2*pi*f*t) + .5*(maxval-
minval)+minval;
    case 'cuadrado'
        x = .5*(maxval-minval) * SQUARE(2*pi*f*t) + .5*(maxval-
minval)+minval;
    case 'triangulo'
        x = min(((rem(t,1/f)/(1/(2*f)))*(maxval-
minval)+minval),((((1/f)-rem(t,1/f))/(1/(2*f)))*(maxval-
minval)+minval));
    otherwise
    x = 0;
end
```

a)

```
>>  [t,x]=generador(0.5,8,'triangulo',-1,0,0,3);
```



b)

```
>>  [t,x]=generador(50000,5000000,'seno',-1,1,0,0.00006);
```