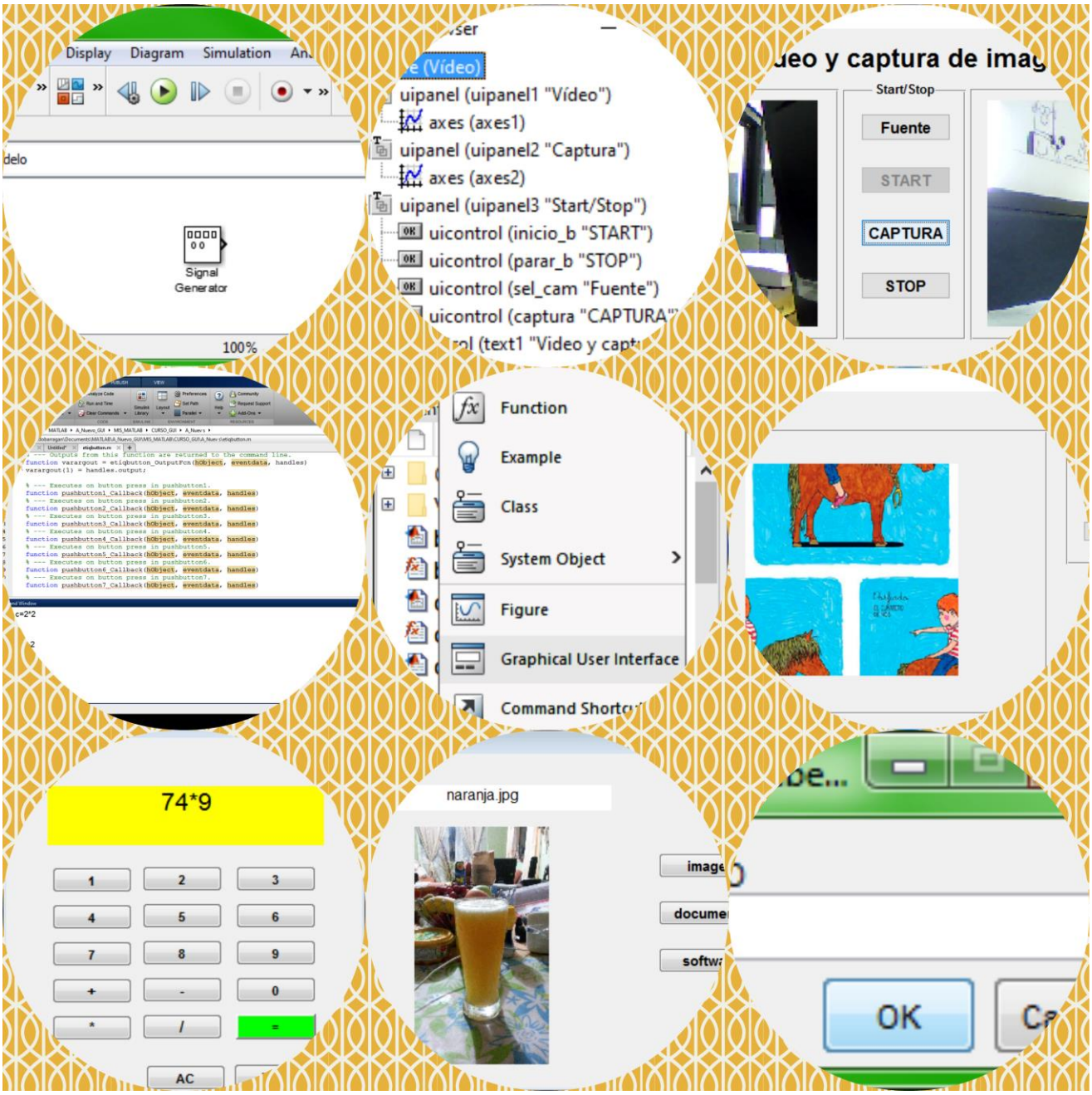

MANUAL GUIDE EN MATLAB

DISEÑANDO INTERFACES GRÁFICAS DE USUARIO

Diego Orlando Barragán Guerrero

GUIDE MATLAB

Diego Barragán Guerrero

Tabla de contenidos

Propósito de este manual

Matlab es uno de los lenguajes más utilizados para aplicaciones de ingeniería. Es usado para simulaciones de sistemas, modelado, graficación de resultados, escritura y lectura de archivos, procesamiento de señales (audio, imágenes y multimedia) y una infinidad de tópicos de ingeniería y tecnologías. Basado en lenguaje texto, varias de estas implementaciones se realizan en archivos denominados *scripts*, funciones o modelos de Simulink. Dado que en algunas de estas implementaciones es necesario un constante ingreso de variables y visualización de resultados, resulta conveniente y necesario crear una interfaz gráfica de usuario (GUI-*graphical user interface*) en la cual se pueda manipular o modificar el ingreso de datos y observar los resultados.

Así, el propósito de este manual es servir como guía introductoria y avanzada en la creación de interfaces gráficas de usuario (GUI) en Matlab. En la primera parte se expone una breve introducción al lenguaje de programación y sus particularidades, propiedades de las funciones, ciclos repetitivos, depuración, etc. A continuación, se explica el uso de las funciones principales de una GUI, como son las funciones *get* y *set*. Finalmente, se presenta una serie de ejemplo de GUI donde se exploran la mayoría de los elementos disponibles como son botones, ventanas de edición de texto, gráficas, etc.

En esta segunda edición del Manual de GUI escrito en 2008, están incluidas múltiples actualizaciones de funciones y propiedades de las interfaces gráficas.

Capítulo 1

Introducción a Matlab.

Matlab se ha consagrado como uno de los lenguajes más populares y versátiles para aplicaciones de ingeniería a nivel mundial. La multiplicidad de funciones, librerías y detallados ejemplos disponibles en la web han hecho accesible el modelado y la implementación de cualquier tipo de sistema, proyecto o algoritmo en este entorno de programación, siendo adoptado en disciplinas o materias relacionadas con ingenierías.

Dado que es uno de los lenguajes más usado y cuya capacidad de procesamiento es cada vez más eficiente, fue incluida hace ya varias versiones la posibilidad de diseñar una interfaz gráfica de usuario (*graphical user interface* - GUI) en la cual sea más sencillo y dinámico el ingreso de datos y la visualización de los resultados, facilitando que los usuarios finales puedan beneficiarse de aplicaciones diseñadas en este lenguaje. En otras palabras, guide (*Graphical User Interface Design*) es un entorno de programación visual disponible en Matlab para realizar programas que necesitan ingreso y visualización continua de datos por parte del usuario final o del programador. Este entorno tiene las características básicas de todos los programas visuales como Visual Basic o Visual C++.

Este manual contiene una guía completa sobre la manera de programar una interfaz gráfica en Matlab, explorando cada uno de los componentes de entrada y salida de datos, así como aplicaciones relacionadas con la adquisición y procesamiento de audio e imágenes que han sido diseñadas y publicadas por el autor desde el 2006 en el *File Exchange* de la web de Matlab (repositorio de intercambio de archivos, el cual contiene miles de programas libres de descarga sobre varias temáticas no solo de ingeniería, sino de otras disciplinas de la ciencia). Finalmente, se presenta una lista de consejos sobre estilo de programación y aceleración de las aplicaciones.

1.1 Breve guía de programación en Matlab.

Antes de iniciar el estudio de programación de GUI, se hará una breve introducción a particularidades de programación en Matlab. Ventanas, operaciones con matrices, formatos de gráficas, propiedades de funciones, entre otros tópicos. En caso que se tenga experiencia en programación de Matlab puede el lector saltar esta sección y pasar directamente a los detalles de la programación de GUI en la sección siguiente. Si no, se recomienda revisar esta sección introductoria realizando y modificando cada uno de los código mostrados en los Listados.

Como recomendación, colocar todos los archivos .m en una sola carpeta y editar el nombre del archivo con un nombre relacionado al ejemplo. También, que el nombre de la carpeta donde están los archivos no contenga espacios o para separar dos nombres usar un guión bajo. Esto, para usar el comando *cd* desde la ventana de comandos para ubicarnos directamente en la carpeta de los programas. Por ejemplo, si grabamos los archivos en la siguiente dirección C:\Users\matlab_gui, usando el comando dentro del *command window* »cd C:\Users\matlab_gui nos ubicará directamente en es carpeta.

1.2 Entorno de Matlab.

El entorno de programación de Matlab está compuesto por un conjunto de ventajas y menús ubicados en la parte superior de la interfaz principal. Cada una de las ventanas pueden ser colocadas en diferentes posiciones por el usuario o eliminar alguna de ellas (por ejemplo, eliminar el *command history*). En la Figura 1.1 se muestra una configuración recomendada de colocación de la ventana de comandos (*commnad window*), la ventana de carpeta actual (*current folder*), la ventana de variables de trabajo (*workspace*) y la ventana con el editor de código (*editor*). En algoritmos donde sea necesario verificar un mayor número de variables presentes en el *workspace*, es recomendable colocar esta ventana en la parte derecha de la pantalla.

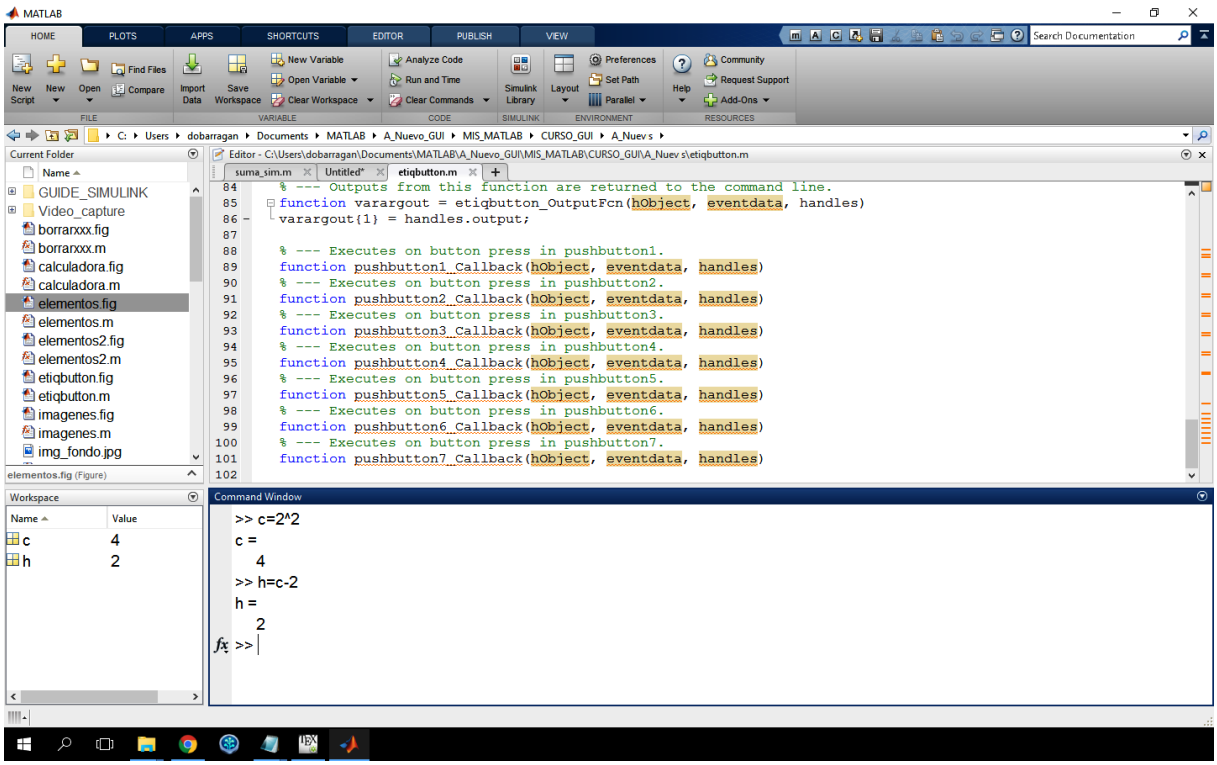

Figura 1.1: Entorno de Matlab.

1.3 Operaciones con matrices.

El Álgebra Lineal define una matriz como un arreglo de dos dimensiones de números. La notación de una matriz A tiene la forma mostrada en (1.1).

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{21} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \tag{1.1}$$

La indexación de cada elemento de la matriz se hace a través del par ordenado (i, j) denotado como $A(i, j)$, donde i corresponde al número de la fila y j al número de la columna. Uno de los aspectos a tomar en cuenta es que Matlab inicia el conteo de los elementos a partir de 1 (uno), que difiere de otros lenguajes como Python o Fortran, donde el conteo inicia en 0.

En Matlab es posible crear matrices desde la ventana de comandos o desde un editor de código. En ambos casos, la matriz creada se verá en el *workspace* o se imprimirá en la ventana de comandos caso la función o comando de creación de la matriz no termine en punto y coma (;). Una vez que la matriz ha sido creada, se puede visualizar sus valores, operarla con otra matriz o bien modificar alguno de ellos haciendo doble clic en la variable correspondiente en el *workspace*.

Como ejemplo, en el Listado 1.1 se crea una matriz de 3 filas por 3 columnas, separando cada fila por un punto y coma (;) y colocando los valores de la matriz dentro de corchetes ([]). A partir de esta matriz, se puede seleccionar cada elemento usando una indexación con el número de fila y el número de la columna, tal cual mostrado en la línea 4 del Listado 1.1. De igual forma, en el código mostrado están las funciones que determinan el tamaño de la matriz y el cálculo de su transpuesta (cambio ordenado de las filas por las columnas). En este ejemplo, el valor de cada elemento de la matriz es un número real, no obstante pueden ser valores fraccionarios o bien número complejos.

Listado 1.1: Creación e indexación de una matriz.

```
1  % Crea una matriz de 3 x 3.
2  M=[1 2 3; ...
3  4 5 6;...
4  8 8 8]
5  % Indexa al elemento de la fila 3 columna 2.
6  em=M(3,2)
7  % Dimensiones de la matriz M.
8  [F,C]=size(M)
9  % Filas de la matriz
10 filas=size(M,1)
11 % Columnas de la matriz
12 columnas=size(M,2)
13 % MT es la matriz traspuesta de M
14 MT=M'
15 [Ft,Ct]=size(MT)
```

Cuando se tienen vectores columna ($m \times 1$) o vectores fila ($1 \times m$) se los puede concatenar para formar una matriz siempre y cuando tengan la misma dimensión (en el caso de un vector fila las dimensiones pueden ser diferentes). A manera de ejemplo, en el Listado 1.2 se crean dos vectores fila que son concatenados uno a continuación de otro (dando como resultado un nuevo vector fila). Si los vectores tienen la misma dimensión, al separarlos por punto y coma (;) se genera una matriz de dimensión 2×3.

Listado 1.2: Concatenación de vectores en una matriz.

```
% Vector fila
v1=[1,1,1]
% Vector fila
v2=[2,2,2]
% Concatenar en un vector fila
v3=[v1,v2]
% Concatenar en una matriz
v4=[v1;v2]
```

Existen una gran cantidad de funciones que manipulan las matrices. Por ejemplo, si se busca repetir con determinada dimensión una matriz se usa la función *repmap*. O si se quiere listar los elementos de un matriz como un vector fila o columna se usa el operador dos puntos (:). (Ver Listado 1.3.)

Listado 1.3: Algunas operaciones con matrices.

```
% repmat
m=[9,8;7,6];
mr=repmat(A,1,4)
% Lista columna
lc=m(:)
% lista fila
lf=m(:)'
```

En algunos casos, cuando se están generando elementos de una matriz dentro de un ciclo repetitivo, es muy recomendable definir de antemano una matriz con elementos todos cero para aumentar la velocidad de procesamiento del algoritmo. La función *zeros* permite crear matrices cuyos elementos son todos cero. Asimismo, la función *ones* devuelve una matriz cuyos elementos son todos uno. (Ver Listado 1.4.)

Listado 1.4: Matrices pre definidas.

```
% Matriz con elementos 0
mat_0=zeros(3,2)
% Matriz con elementos 1
mat_1=ones(2,3)
% Matriz con elementos 5
mat_5=5*ones(2,3)
```

En algunas aplicaciones, se requiere trabajar con determinados elementos de una matriz, sea una fila o una columna o parte de ella. Extraer una fila, una columna o una sección de una matriz es posible gracias al operador dos puntos (:). Este operador se puede leer como *todos los elementos*. (Ver Listado 1.5.) De igual forma, la función *end* permite indicar que se requiere extraer los datos de la matriz hasta el final de su dimensión, es decir, *M(1,3:end)* tomará de la fila uno (1) los elementos desde la columna 3 hasta la última columna de esa fila.

Listado 1.5: Matrices pre definidas.

```
% Matriz de 4x4
M=[.1, .2 ,.3  ,.4;
   .5, .6 ,.7  ,.8;
   .9, .11,.12 ,.13;
```

```
   .14,.15,.16  ,.17];
% Todos los elementos de la columna 2
M(:,2)
% Todos los elementos de la fila 3
M(3,:)
% Elementos de la fila 1 a la 3 y
% de la columna 2 a la 4
M(1:3,2:4)
% Eliminar toda la fila 1
M(1,:)=[]
% Agregar una columna de 1
M(:,5)=ones(3,1)
% end
M(1,3:end)
```

1.4 Gráficas.

Matlab permite realizar gráficas en dos y tres dimensiones, permitiendo también editarlas con texto, flechas y leyendas descriptivas. La función para realizar gráficas 2D en Matlab con los datos de vectores es la función *plot* para señales continuas y la función *stem* para señales discretas o secuencias discretas. Esta función acepta varios argumentos para parametrizar la gráfica (color, tamaño y tipo de línea, etc). Como ejemplo inicial, se graficará una función cuadrática con el código del Listado 1.6.

Listado 1.6: Función *plot*.

```
% Vector x
x=1:10;
% Cuadrado del vector x
fc=x.^2;
% Figura de fc
plot(x,fc)
```

Como resultado se obtendrá una nueva ventana con una figura cuya función es el cuadrado de valores entre uno a diez. (Ver Figura 1.2a.) Notar que Matlab incluye una etiquetación por defecto para el espacio entre los valores de x y y, un color azul y un ancho de línea continua de valor 1.

A partir de este código, es posible añadir más características a la figura de esta función cuadrática. Colocar un título, etiqueta del eje x y del eje y se lo hace con el código del Listado 1.7. Todo el código relacionado con la modificación de característica visuales de la figura debe ir después de la función *plot*. Caso contrario, el compilador no dará error, mas los cambios no se verán en la figura. El resultado se muestra en la Figura 1.2b.

Listado 1.7: Título y etiquetas de una gráfica.

```
% Vector x
x=1:10;
% Cuadrado del vector x
fc=x.^2;
% Figura de fc
plot(x,fc)
```

```
% Titulo de la figura
title('Funcion cuadratica')
% Etiquetas de los ejes x, y
xlabel('Eje x')
ylabel('x^2')
```

En algunos casos, para mejorar la visualización de una cierta parte de la gráfica, es necesario modificar hasta donde se muestran los datos en la figura, es decir, los límites de x y y. Esto se lo hace con el código del Listado 1.8. Asimismo, se puede añadir una rejilla con la función (*grid*).

Listado 1.8: *Grid* y límites de la figura.

```
% Vector x
x=1:10;
% Cuadrado del vector x
fc=x.^2;
% Figura de fc
plot(x,fc)
% Titulo de la figura
title('Funcion cuadratica')
% Etiquetas de los ejes x, y
xlabel('Eje x')
ylabel('x^2')
% Rejilla
grid on
% Limites de la figura
xlim([2,8])
ylim([4,64])
```

En la Figura 1.2c se puede observar como los límites x y y han sido modificados, así como la rejilla o *grid* que fue añadida a la Figura.

Dentro de la misma ventana de la Figura, se pueden graficar varias funciones matemáticas (o señales), usando la función *hold*. En el código del Listado 1.9 se usa una configuración adicional en la segunda función *plot* con el fin de modificar el color de la señal a rojo. La Figura 1.2d muestra este resultado.

Listado 1.9: Dos gráficas en una misma figura con la función *hold*.

```
% Vector x
x=1:10;
% Cuadrado del vector x
fc=x.^2;
% Figura de fc
plot(x,fc)
% Titulo de la figura
title('Funcion cuadratica')
% Etiquetas de los ejes x, y
xlabel('Eje x')
ylabel('x^2')
% Rejilla
grid on
% Limites de la figura
xlim([2,8])
```

```
16 ylim([4,64])
17 % Dos graficas en una misma figura
18 hold on
19 plot(x,2*fc,'Color','r')
20 hold off
```

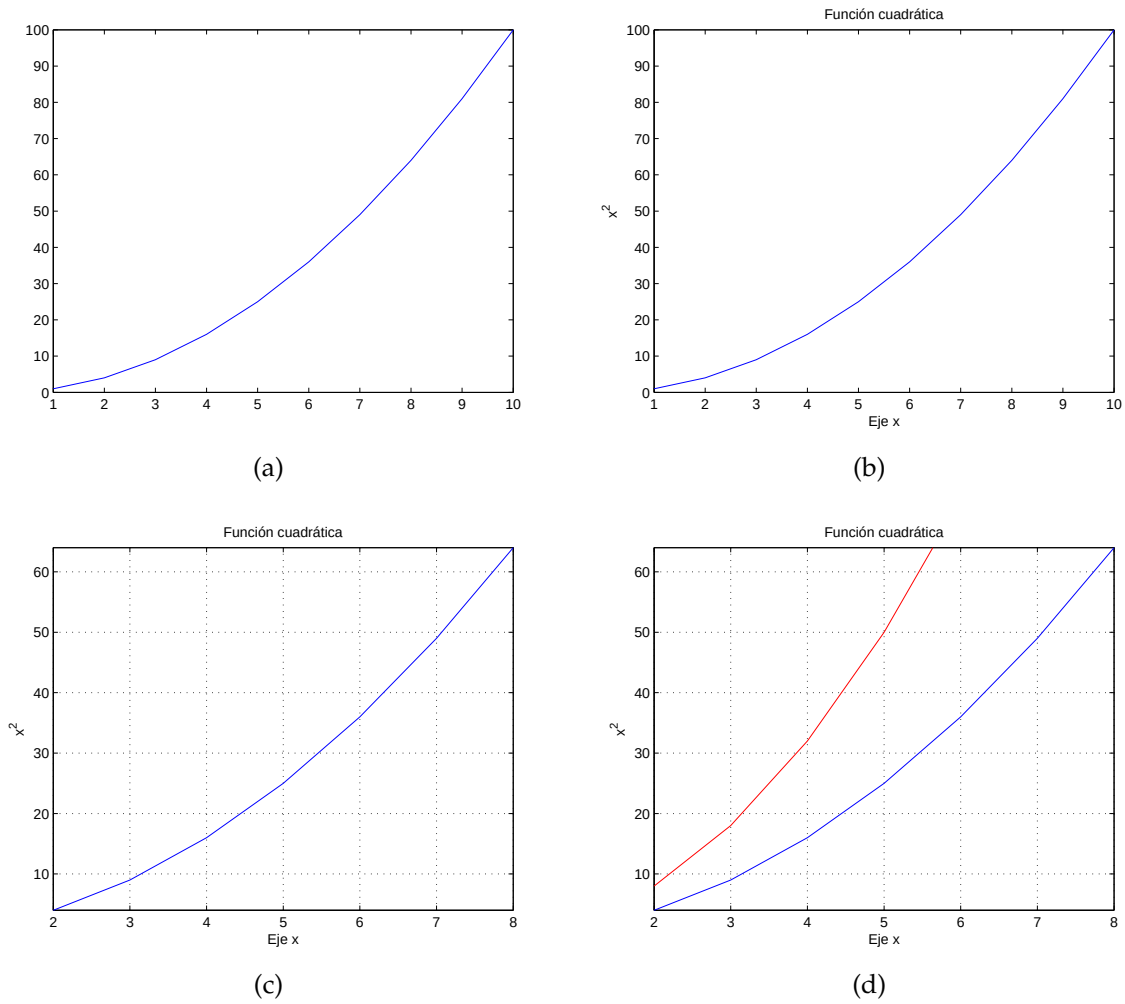

Figura 1.2: (a) *Plot* simple. (b) *Plot* con etiquetas. (c) *Plot* con grid. (d) *Plot* con *hold*.

Para graficar ambas señales en dos ventanas diferentes se coloca la función *figure* antes del segundo *plot*. Y si se necesita graficar ambas señales en dos espacios diferentes pero dentro de la misma ventana de la figura, se emplea la función *subplot*. La función *subplot* tiene como argumento una matriz. Por ejemplo, si vamos a graficar seis figuras en una sola ventana, podemos ordenarlas como una matriz de dos filas y tres columnas, usando la configuración de la Figura 1.3.

subplot(2,3,1)	subplot(2,3,2)	subplot(2,3,3)
subplot(2,3,4)	subplot(2,3,5)	subplot(2,3,6)

Figura 1.3: Uso de la función *subplot*: orden de las gráficas.

1.5 Funciones y *scripts*.

Existen dos tipos de archivos .m: funciones y *scripts*. Una función en un conjunto de código en el cual es necesario ingresar determinados parámetros y para conseguir ciertos parámetros de salida. Podemos pensar que un ejemplo de función es un código que calcula la raíz cuadrada de un número donde el parámetro de entrada es el número y el parámetro de salida es la raíz cuadrada de éste. Ya un *script* es un código donde los parámetros de entrada están definidos dentro del mismo programa.

La importancia de las funciones radica en que al estar programadas en archivos aparte o fuera del código principal, se las puede llamar en cualquier momento ahorrando repetición de código y ganando claridad en la programación. También, estas funciones pueden contener dentro de ellas otras funciones, es decir, funciones anidadas. Uno de las cuestiones a tener en cuenta en las funciones es que el nombre de la función dentro del archivo .m debe ser el mismo nombre con el cual grabamos el archivo en disco.

Para entender mejor la diferencia entre estos tipos de archivo, inicialmente, se creará un *script* para representar un fenómeno de amortiguamiento. Luego, se convertirá este código *script* en una función que será llamada desde otro archivo .m.

Listado 1.10: *Script* del fenómeno de amortiguamiento.

```
% Vector de tiempo
t=0:1/1000:1.5;
% Exponencial decreciente
exp_fcn=exp(-t);
% Seno de 10 Hz (T=0.1 s)
sin_fcn=sin(2*pi*10*t);
% Multiplicar ambas funciones
am_fcn=exp_fcn.*sin_fcn;
% Graficar
plot(t,am_fcn)
% Mantener la figura anterito
hold on
% Graficar la envolvente exponencial
plot(t,exp_fcn,'r--')
plot(t,-exp_fcn,'r--')
hold off
```

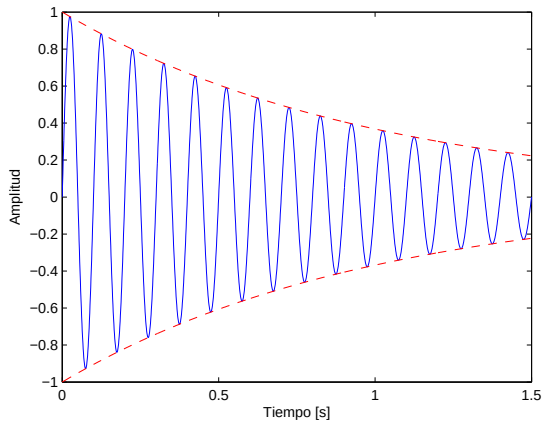

Figura 1.4: Representación del fenómeno del amortiguamiento.

En el *script* del Listado 1.10, varios parámetros pueden ser modificados dentro del programa. Por ejemplo: la frecuencia de la onda seno, la velocidad de decaimiento de la función exponencial, la amplitud de ambas funciones, el color de las funciones graficadas, etc. Al convertir este *script* en una función, se modificarán los parámetros de frecuencia de la onda seno y la velocidad de amortiguamiento de la señal exponencial decreciente. El resto de parámetros quedarán sin modificación. (Ver Listado 1.11.)

Listado 1.11: Función del fenómeno de amortiguamiento.

```
function fcn_amort(frec_seno,expo_deca)
% Vector de tiempo
t=0:1/1000:1.5;
% Exponencial decreciente
exp_fcn=exp(-expo_deca*t);
% Seno de 10 Hz (T=0.1 s)
sin_fcn=sin(2*pi*frec_seno*t);
% Multiplicar ambas funciones
am_fcn=exp_fcn.*sin_fcn;
% Graficar
plot(t,am_fcn)
% Mantener la figura anterior
hold on
% Graficar la envolvente exponencial
plot(t,exp_fcn,'r--')
plot(t,-exp_fcn,'r--')
xlabel('Tiempo [s]')
ylabel('Amplitud')
hold off
```

Para probar el resultado de esta función, desde la ventana de comandos ejecutamos cualquiera de los comandos del Listado 1.12. En la misma ventana de comandos se verán también valores de salida de la función, caso dentro de esta estén programadas.

Listado 1.12: Prueba de la función del fenómeno de amortiguamiento.

```
>> fcn_amort(10,1)
>> fcn_amort(11,1)
>> fcn_amort(12,1)
>> fcn_amort(20,1)
>> fcn_amort(20,2)
```

1.6 Funciones de control de flujo: *if*, *switch-case*, *for*, *while*.

Las funciones *if-elseif-end* y *switch-case* se usan cuando se necesita ejecutar una sección de código dependiendo de un resultado anterior o comparación de resultados. En cambio las funciones *for* y *while* se utilizan para repetir varias veces un mismo código. En el Listado 1.13 se muestra un ejemplo del uso de *if*. La función *disp* presentará un mensaje en la ventana de comandos.

Listado 1.13: Función if-else.

```
% Valor aleatorio entre 0 y 1
n=rand(1);
% Ver si el valor es mayor que 0.5
```

```
if n<0.5
    disp('Valor debajo de 0.5')
% Caso contrario:
else
    disp('Valor superior a 0.5')
end
```

Para añadir más comparaciones se lo realiza con la función *elseif*. En el Listado 1.14 se muestra unas sentencias con varias comparaciones donde se usa la operación lógica *and* (puede probarse el código cambiando la operación *and* por una *or*). Se pueden añadir tantas comparaciones con *elseif* como sean necesarias. No obstante, tener presente que la última comparación se hace con la función *else*.

Listado 1.14: Función *if-elseif-else*.

```
% Valor aleatorio entre 0 y 1
n=rand(1);
% Ver si el valor es mayor que 0.5
if n<0.5 && n>0.3
    disp('Valor en el rango 0.3-0.5')
elseif n<1 && n>0.6
    disp('Valor en el rango 0.6-1.0')
% Caso contrario:
else
    disp('Valor fuera de los rangos')
end
```

En el Listado 1.14 se usó tanto una comparación de menor y mayor (< >) así como una función de comparación lógica (&&), la operación *and*. Estas operaciones sirven para comparar valores numéricos, así como cadenas de texto. La función *swithc-case* es semejante a la función *if-elseif*. El listado 1.15 se muestra un ejemplo donde se compara un *string*. Al igual que *if-elseif*, se puede agregar tantas comparaciones como sean necesarias recordando que la última comparación va con el comando *otherwise*.

Listado 1.15: Función *switch-case*.

```
n='uno';
switch n
    case {'uno','dos'}
        disp('Uno o Dos')
    case {'tres', 'cuatro'}
        disp('Tres o Cuatro')
    case 'cinco'
        disp('Cinco')
    otherwise
        disp('Otro valor')
end
```

Una de la características que diferencia a la función *switch-case* de la función *if-elseif* es que la primera admite comparaciones de variables (por ejemplo, comparaciones lógicas entre escalares: <> o &&) pero no las ejecuta de forma correcta. En sí, el código no presentará ningún error de compilación, pero el resultado no será el esperado. En el Listado 1.16 se muestra un ejemplo de este caso particular.

Listado 1.16: Función *switch-case* con resultado incorrecto.

```
n=3
switch n
    case n>5
        disp('n es mayor que 5')
    case n>1 && n<4
        disp('n mayor que 1 y menor que 4')
    otherwise
        disp('Otro valor')
end
```

Con $n = 3$, el valor que se presenta en la ventana de comandos debería ser *n mayor que 1 y menor que 4*. No obstante, el resultado será la función del código *otherwise*. En resumen, usar la función *if-elseif* siempre que se quiera ejecutar código en base a la comparación de valores. Usan *switch-case* cuando la comparación sea hacia un solo valor o se necesite comparar entre varios *strings*.

En cuanto a las funciones para crear ciclos repetitivos, tanto *for* como *while* cumplen un rol semejante. No obstante, *for* se recomienda usar para ciclos repetitivos fijos, mientras que *while* para ciclos repetitivos sujetos a una condición externa (por ejemplo, hasta que cierto valor pase de un umbral). Ambas funciones admiten un par de comando tanto para detener por completo el ciclo repetitivo o bien saltar a la siguiente iteración. Estos comando son *brake* y *continue*. En el Listado 1.17 se presenta un ejemplo de estas funciones.

Listado 1.17: Ciclos iterativos for y while.

```
%% Ejemplo de for
for conta=1:10
    disp(num2str(conta))
end
%% Ejemplo de while
conta2=0;
while conta2<10
    disp(num2str(conta2))
    conta2=conta2+1;
end
%% Funciones continue y brake
conta2=0;
while conta2<10
    conta2=conta2+1;
    if conta2==5 %No se imprime el 5
        continue
    end
    disp(num2str(conta2))
    if conta2==8 % El ciclo se detiene en 8 conteos
        break
    end
end
```

1.7 Creación de atajos.

Durante la programación en Matlab puede ser necesario en ciertos casos cerrar todas las ventanas que contienen una figura, borrar la ventana de comandos o eliminar todos los datos del *workspace*. Todo esto se lo hace con funciones como *clc*, *clear all* y *close all*. Otra necesidad que puede acontecer durante la programación es la de abrir la carpeta que contiene los códigos desarrollados. El comando *pwd* ejecutado en la ventana de comandos nos da la ruta del directorio actual. Junto con la función *winopen* se consigue abrir en el explorador el directorio de trabajo con la combinación *winopen(pwd)*.

Asimismo, resulta cómodo con un solo comando ubicarse en la carpeta donde se está desarrollando los códigos almacenados en archivos .m. Esta ubicación de la carpeta del proyecto en el *curren folder* se consigue con el comando *cd* y la dirección en disco de la carpeta. Matlab dispone de una herramienta bastante útil para ejecutar cada una de estas funciones con un solo clic. Esta herramienta son los atajos, que se pueden crear de forma fácil haciendo clic en el ícono *New shortcut*, con lo que aparecerá la ventana mostrada en la Figura 1.5

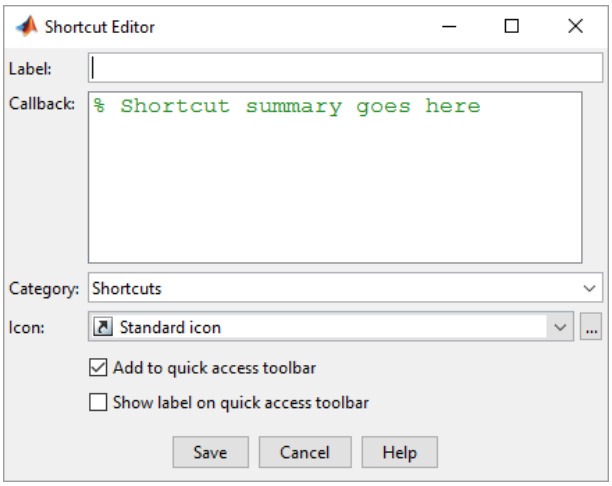

Figura 1.5: Ventana de configuración de atajo.

Resta colocar en la venta *Callback* los comandos que se ejecutarán cada vez que se haga clic en el nuevo ícono. Por ejemplo, el atajo para borrar todas las variables, figuras y ventana de comandos se muestra configurado en la Figura 1.6. De esta forma resultará cómodo con un solo clic tener borradas todas las variables y cerrado todas las gráficas para iniciar una nueva simulación.

Los atajos creados pueden observarse en la parte superior derecha del entorno de Matlab, conforme mostrado en la Figura 1.7.

1.8 Uso del *debug*: ejecutar línea a línea.

Durante la programación o implementación de un algoritmo, muchas veces es necesario conocer el valor que adquiere una variable dentro del programa a cada iteración. Por ejemplo, saber los datos que se generan dentro de un ciclo repetitivo o conocer si el programa entra o no dentro del código de una función *if*. Es en estos casos donde la utilización del modo *debug* de Matlab facilita de gran manera el trabajo de analizar y observar los valores que adquiere cada variable durante la simulación. Como ejemplo de uso del depurador, vamos a verificar valor a valor el resultado del cálculo

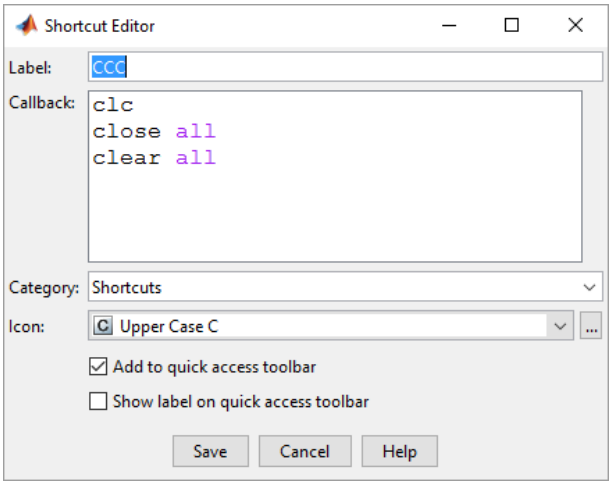

Figura 1.6: Configuración del atajo de borrado.

Figura 1.7: Atajos creados para borrar todo, abrir carpeta actual y abrir carpeta de proyecto.

de la serie de Fibonacci mostrada en el Listado 1.18.

Listado 1.18: Serie de Fibonacci.

```
n=11;
sf=zeros(1,n);
sf(1) = 1;
sf(2) = 1;
for i = 3 : n
    previo_1=sf(i-1);
    previo_2=sf(i-2);
    sf(i) = sf(i-1) + sf(i-2);
    pr_fib=sf(i);
end
disp(['Los ',num2str(n),' primeros valores de la serie de
   Fibonacci: '])
sf
```

Para colocar un punto de parada (o *break point*) basta con dar clic en la línea horizontal que está cerca a la numeración del *script* (ver Figura 1.8). Esto creará un punto rojo que indica que en esa línea el programa se detendrá y podremos ver el *workspace* el valor de todas las variables generadas hasta ese momento por el programa.

Luego, al ejecutar F5 para correr el programa aparecerá una flecha verde en la línea donde se colocó el punto de parada y una opción de depuración en el menú superior con el cual podemos ejecutar línea por línea (o con la tecla F10), correr el programa hasta el cursor o bien entrar a una función y ver detalles de cálculos de la misma. Durante esta depuración, podemos observar el valor que adquiere cada variable en el *workspace*.

De esta manera, si se observa el *workspace*, se verán los valores que adquieren las variables en cada iteración, consiguiendo así un mejor control sobre el desarrollo del programa, verificando errores y modificando cualquier problema.

```
n=11;
sf=zeros(1,n);
sf(1) = 1;
sf(2) = 1;
for i = 3 : n
    previo_1=sf(i-1);
    previo_2=sf(i-2);
    sf(i) = sf(i-1) + sf(i-2);
    pr_fib=sf(i);
end
disp(['Los ',num2str(n),' primeros términos de la serie de Fibonacci: '])
sf
```

Figura 1.8: *Break point* para ejecutar en modo *debug*.

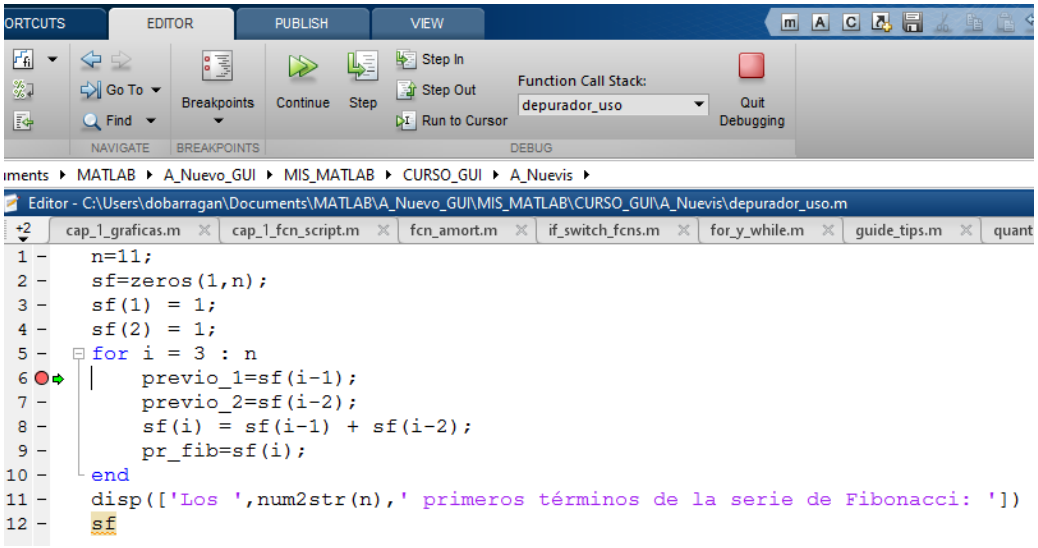

Figura 1.9: *Break point* para ejecutar en modo *debug*.

1.9 Escritura y lectura de datos en archivos de texto.

Los datos procesados por Matlab pueden ser escritos en un archivo de texto (extensión .txt) para ser exportados a otro paquete de software o leídos en cualquier editor de texto. Las funciones necesarias son *fopen*, *fprintf* y *fclose*. Con la primera función, se define el nombre y extensión del archivo de texto así como también si este archivo va a ser leído o escrito. La función *fprintf* define el formato y la cadena de texto a ser escrita en el archivo. Al finalizar la escritura, se cierra el archivo usando la función *fclose*. El uso de esta última función es necesaria para poder manipular el archivo de texto creado sin que exista conflicto con el sistema operativo (es decir, si no se cierra el archivo desde Matlab, no será posible borrar el archivo o cortarlo y pegarlo en otra ubicación del disco duro).

En el Listado 1.19 se muestran las funciones necesarias para crear y editar un archivo de texto desde Matlab. Notar que al ejecutar este código se creará el archivo en la misma carpeta donde está ubicado el *script*. No obstante, si se desea grabar el archivo en otra ubicación en el disco, basta con añadir la ruta completa antes del nombre del archivo en la función *fopen*.

Listado 1.19: Escritura de archivo de texto.

```
% Crea el archivo de texto para escritura
fid = fopen('texto_salida.txt', 'wt');
% Escritura de datos
fprintf(fid,'%s\n','Nombre:');
```

```
fprintf(fid,'%s\n',num2str(1234));
fprintf(fid,'%s\n','Apellido: ');
fprintf(fid,'%s\n',num2str(5678));
fprintf(fid,'%s\n','DNI: ');
fprintf(fid,'%s\n',num2str(118956445));
fprintf(fid,'%s\n','Nacimiento: ');
fprintf(fid,'%s\n',date);
% Cerrar escritura del archivo
fclose(fid);
% Opcional
winopen('texto_salida.txt')
```

En el Listado 1.19, la función *fopen* crea el archivo texto_salida.txt en modo de escritura (parámetro *wt*). La función *fprintf* escribe en formato *string* (cadena de texto) el texto y además da un salto de línea con el comando \n. Puede añadirse tantas líneas como sean necesarias copiando y pegando este comando. Aparte del formato *string*, esta función *fprintf* puede escribir un número con tantos decimales como sean necesarios o también en formatos tipo punto fijo, notación exponencial, doble precisión, etc.

Como ejemplo de lectura de archivos de texto, leeremos datos que correspondan a una variable x y y para que sean graficados con la función *plot*. Consideremos el archivos de texto mostrado en la Figura 1.10.

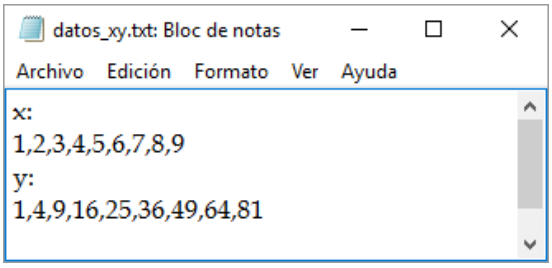

Figura 1.10: Archivo de texto a ser leído en Matlab.

Como muestra la Figura 1.10, el archivo de texto contiene cuatro líneas, siendo la primera y la tercera etiquetas y la segunda y cuarta los datos que serán graficados. En Matlab, la lectura de este archivo es algo semejante a la escritura ya que primero creamos un identificador del archivo con la función *fopen*, indicando que el archivo será leído (parámetro *r*). La función *fscanf* lee los datos y con la función *find* encontramos la posición de los dos puntos (:) con los cuales segmentamos los datos de x y de y. Ya para graficar, convertimos los datos de formato texto (string) a dato numérico con la función *str2num*. El Listado 1.20 contiene el código para la lectura de datos, cambio de formato y graficación.

Listado 1.20: Lectura de archivo de texto.

```
% Crear id del archivo de texto.
identi = fopen('datos_xy.txt','r');
% Leer los datos del arhivo.
A = fscanf(identi,'%s');
% Encontrar los : para segmentar x,y.
pdp=find(A==':');
% Segmentar x.
```

```
x=A(pdp(1)+1:pdp(2)-2);
% Segmentar y.
y=A(pdp(2)+1:end);
% Convertir a vector
x_num=str2num(x);
y_num=str2num(y);
% Graficar
plot(x_num,y_num)
```

1.10 *Save* y *open*.

Cada uno de los resultados procesados por Matlab se guardan en una variable temporal dentro del *workspace* que luego puede sea leída o modificada. Si se ejecuta el comando de borrado *clear all* todas las variables del *workspace* serán eliminadas. Al usar la función *save* se guarda en disco los datos contenidos en una de las variables del *workspace* en un archivo con extensión .mat. La sintaxis de esta función se presenta en el Listado 1.21. El primer parámetro de la función es el nombre del archivo que se creará en disco. A continuación, el nombre de las variables que serán guardadas en este archivo.

Listado 1.21: Guardar variables en disco con la función save.

```
m = [1,2,3,4,5];
t = m';
save('a_disco.mat','m','t')
```

Una vez ejecutado el código del Listado 1.21, leer los datos contenidos en este nuevo archivo se lo hace con la función *load*: *load('a_disco')*. Así, las variables almacenadas estarán nuevamente dentro del *workspace* para ser leídas o modificadas.

Capítulo 2

Creación de una GUI.

Iniciar un nuevo proyecto en GUI es sencillo. Se tiene dos formas para hacerlo. La primera consiste en escribir *guide* en el *command window* y presionar la tecla enter. La segunda forma consiste en seleccionar la opción que se muestra en la Figura 2.1.

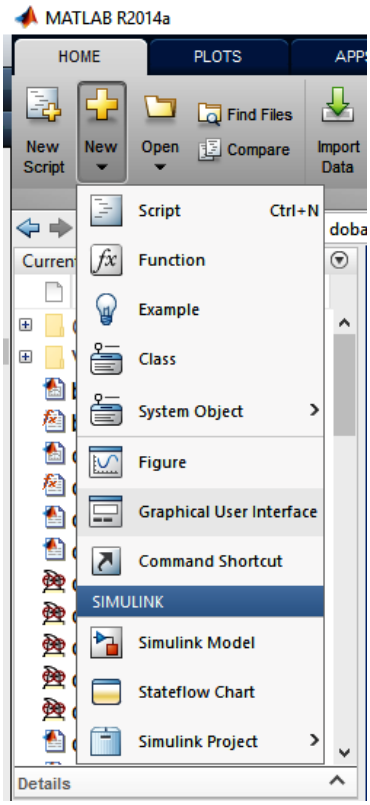

Figura 2.1: Nuevo proyecto GUI.

En ambos casos, se presentan cuatro opciones en la ventana de inicio de la GUI (Figura 2.2). Estas son:

a. **Blank GUI (Default)**: La opción de interfaz gráfica de usuario en blanco (viene predeterminada), nos presenta un formulario nuevo, en el cual podemos diseñar nuestro programa.

b. **GUI with Uicontrols**: Esta opción presenta un ejemplo en el cual se calcula la masa, dada la densidad y el volumen, en alguno de los dos sistemas de unidades. Podemos ejecutar este ejemplo y obtener resultados.

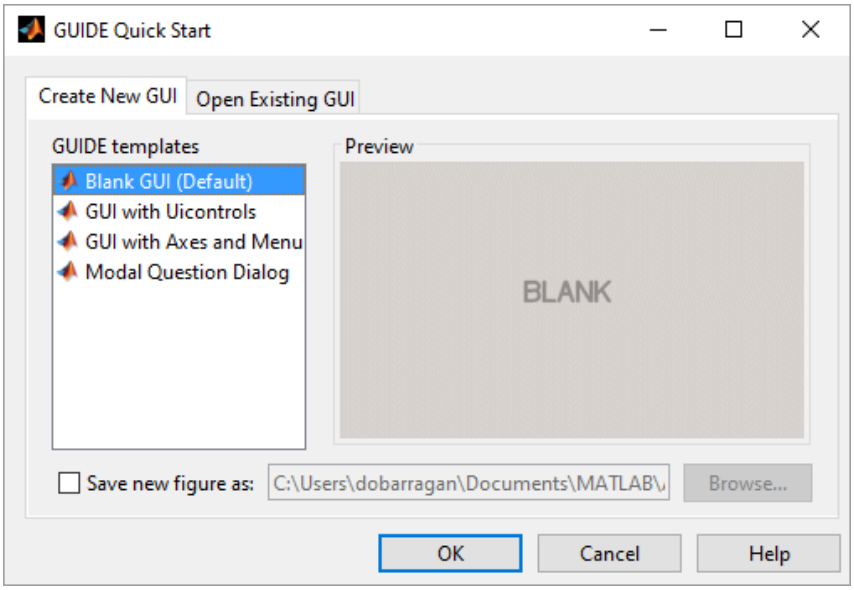

Figura 2.2: Ventana de inicio de GUI.

c. **GUI with Axes and Menu**: Esta opción es otro ejemplo el cual contiene el menú File con las opciones Open, Print y Close. En el formulario tiene un Popup menu, un push button y un objeto Axes, podemos ejecutar el programa eligiendo alguna de las seis opciones que se encuentran en el menú despegable y haciendo click en el botón de comando.

d. **Modal Question Dialog**: Con esta opción se muestra en la pantalla un cuadro de diálogo común, el cual consta de una pequeña imagen, una etiqueta y dos botones Yes y No, dependiendo del botón que se presione, el GUI retorna el texto seleccionado (la cadena de caracteres 'Yes' o 'No').

Se elige la primera opción (Blank GUI) y se tendrá la ventana que se presenta en la Figura 2.3.

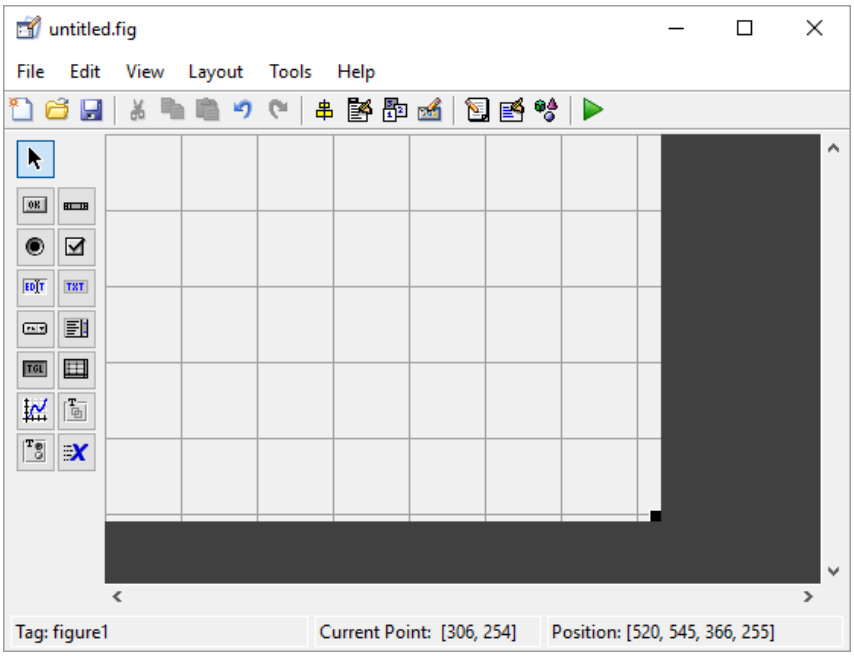

Figura 2.3: Entorno de programación de GUI.

Para obtener la etiqueta en cada elemento de la paleta de componentes se ejecuta: *File»Preferentes* y seleccionamos *Show names in component palette*. El entorno de diseño se modifica como muestra la Figura 2.4.

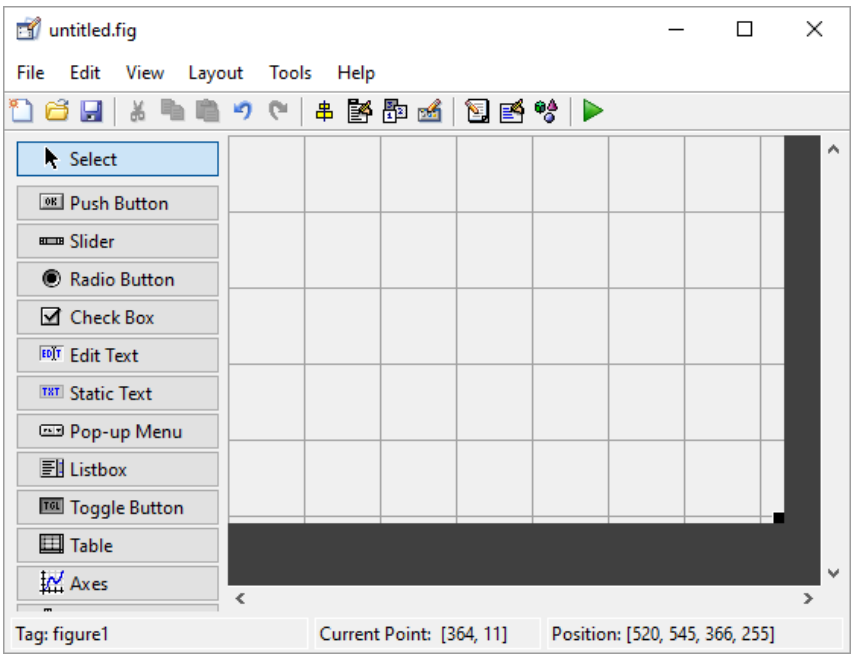

Figura 2.4: Entorno de programación de GUI con elementos etiquetados.

2.1 Propiedades de los componentes.

Cada uno de los elementos de la paleta de componentes tiene un conjunto de opciones o propiedades que se acceden con el clic derecho conforme mostrado en la Figura 2.5.

La opción *Property Inspector* (Figura 2.6) permite personalizar cada componente, como por ejemplo cambiar el tipo y tamaño de letra, color, etiqueta, posición y reacción del componente a clics del usuario. En esta ventana podemos modificar el nombre (campo *tag*) con el cual este componente aparecerá en el archivo .m asociado.

Al hacer clic derecho en el componente ubicado en el área de diseño, una de las opciones más importantes es *View Callbacks*, la cual, al ejecutarla, abre el archivo .m asociado a nuestro diseño y nos posiciona en la parte del programa que corresponde a la función que se ejecutará cuando se realice una determinada acción sobre el componente que estamos editando. Por ejemplo, al ejecutar *View Callbacks»Callbacks* en el *Push Button*, nos ubicaremos en la parte del programa mostrado en el Listado 2.1.

Listado 2.1: Función del push button

```
1 % --- Executes on button press in pushbutton1.
2 function pushbutton1_Callback(hObject, eventdata, handles)
3 % hObject    handle to pushbutton1 (see GCBO)
4 % eventdata  reserved - to be defined in a future version of
     MATLAB
5 % handles    structure with handles and user data (see
     GUIDATA)
```

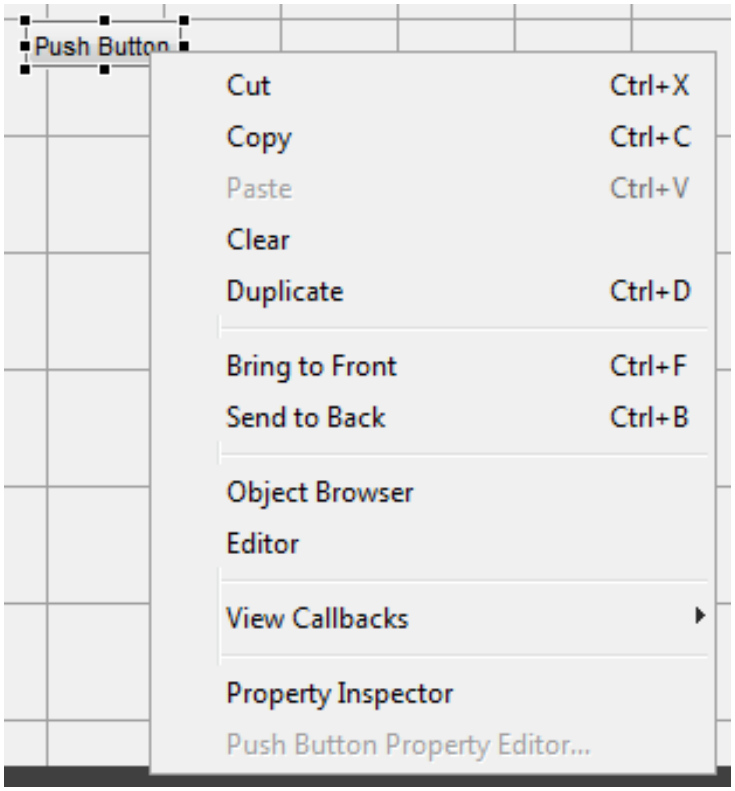

Figura 2.5: Opciones del componente *push button*.

2.2 Archivos y funcionamiento de una GUI.

Una GUI en Matlab consta de dos archivos con extensiones .m y .fig. El archivo .m contiene el código o funciones correspondientes a los componentes de la interfaz. El archivo .fig contiene los elementos gráficos. Cada vez que se adicione un nuevo componente en la interfaz gráfica, se genera de forma automática el código en el archivo .m. Para ejecutar una interfaz gráfica, si se la etiquetó, por ejemplo, con el nombre curso.fig, simplemente ejecutamos en la ventana de comandos » curso. O haciendo clic derecho en el archivo .m y seleccionando la opción *run*.

2.3 Manejo de datos entre los componentes gráficos de la GUI y el archivo asociado .m

Todos los valores de las propiedades del componentes (color, valor, posición, etiqueta, etc.) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son accedidos mediante un único y mismo identificador para todos éstos. Tomando el programa listado anteriormente, el identificador se asigna en *handles.output = hObject*.

La variable *handles* es nuestro identificador a los datos de la aplicación. Esta definición de identificador es actualizada con la siguiente instrucción: *guidata(hObject, handles);*. La variable *guidata* guarda los datos de la función asociada al componente. Como regla general, en cada función se debe escribir en la última línea la siguiente sentencia: *guidata(hObject, handles);*. Esta sentencia nos garantiza que cualquier cambio o asignación de propiedades o variables quede almacenado.

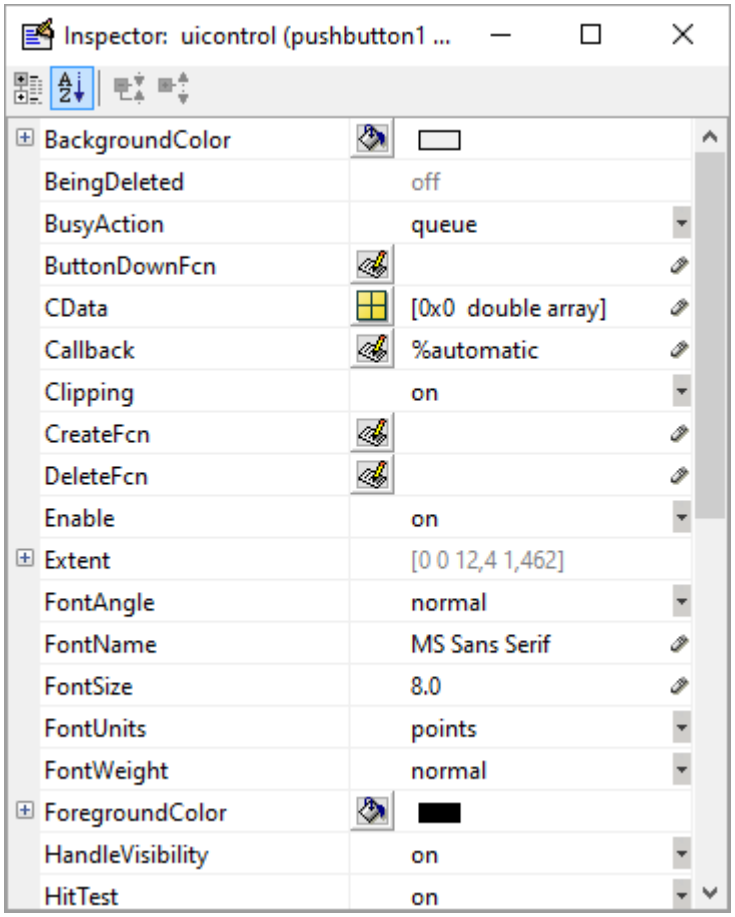

Figura 2.6: Entorno *Property Inspector*. Permite ver y editar las propiedades de un componente.

Por ejemplo, si dentro de una función cierta operación dio como resultado una variable llamada *valor*, para poder utilizarla desde el programa u otra función se debe guardar de la manera indicada en el Listado 2.2:

Listado 2.2: Uso de *guidata*: exportar datos a otra función.

```
1 handles.valor=valor;
2 guidata(hObject,handles);
```

En la primera línea se ingresa la variable *valor* en la estructura de datos de la aplicación apuntada por *handles*. En la segunda, se almacena el valor. De esta manera, en cualquier otra función del archivo .m asociado, se puede leer esta variable conforme el código indicado en el Listado 2.3.

Listado 2.3: Uso de *guidata*: importar datos de otra función.

```
1 valor=handles.valor;
```

2.4 Funciones *get* y *set*.

La asignación u obtención de valores de los componentes se realiza mediante las funciones *get* y *set*. Por ejemplo, si queremos que la variable *dato* tenga el valor del componente *slider* etiquetado como *slider1*, se escribe el código de Listado 2.4.

Listado 2.4: Sintaxis de la función get.

```
1 dato= get(handles.slider1,'Value');
```

Notar que siempre se obtienen los datos a través de los identificadores *handles*. Para asignar el valor a la variable *datos* al componente *statictext* etiquetado como *text1* se escribe el código del Listado 2.5.

Listado 2.5: Sintaxis de la función set.

```
1 set(handles.text1,'String',datos);
```

Capítulo 3

Ejemplos de GUI

Esta sección contiene varios ejemplos de programación de interfaces gráficas de usuario (*graphical user interface* - GUI) usando cada uno de los componentes disponibles en Matlab. En la mayoría de cada uno de los ejemplos se presenta el código completo junto con comentarios de la programación de las funciones asociadas a cada componente.

3.1 GUI de presentación de un diseño.

En ciertos diseños es necesario colocar una breve presentación de las características del programa o una información introductoria de la GUI. En el siguiente código se muestra como crear una pantalla previa a la ejecución del programa principal que contiene una imagen de fondo, información del programa y un botón que abre la GUI siguiente, que es el programa principal. La Figura 3.1 muestra el entorno de este programa de presentación.

Figura 3.1: Pantalla de presentación inicial.

Colocamos el código de la Lista 3.1 en un nuevo archivo de Matlab. La GUI resultante

se muestra en la Figura 3.1.

Listado 3.1: Código de GUI de presentación del programa principal.

```
function presentacion
clear,clc,cla,close all
%Creamos figura
figdiag=figure('Units','Pixels',...
                    'Position',[0.0725 0.0725 600 400],... %
                       Tamano
                        'Number','off',...
                'Name','Manual de GUI en Matlab', ...
                'Menubar','none', ...
                'color',[0 0 0]);

%Ubicamos ejes en figura
axes('Units','Normalized',...
     'Position',[0 0 1 1]);
 %-----Centramos la figura---------
scrsz = get(0, 'ScreenSize');
pos_act=get(gcf,'Position');
xr=scrsz(3) - pos_act(3);
xp=round(xr/2);
yr=scrsz(4) - pos_act(4);
yp=round(yr/2);
set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);
%--------------------------------------
%Incluir imagen
%Importamos imagen *.jpg,junto con su mapa de colores
[x,map]=imread('presentacion_img.jpg','jpg');
%Representamos imagen en figura, con su mapa de colores
image(x),colormap(map),axis off,hold on

%Titulos sobre imagen
%Titulo
text(50,180,'Presentacion del Programa','Fontname','Arial',
   ...
    'Fontsize',25,'Fontangle','Italic','Fontweight','Bold',
       ...
    'color',[0.8 1 0.8]);

%Nombre del programador
text(50,330,'Por: Diego Barragan Guerrero','Fontname','Comic
   Sans MS',...
    'Fontangle','Italic','Fontweight','Bold','Fontsize',14,
       ...
    'color',[1 0.5 0.5]);

%Boton Continuar: abre la GUI siguiente.
botok=uicontrol('Style','pushbutton', ...
        'Units','normalized', ...
    'Position',[.74 .03 .22 .09], ...
        'String','CONTINUAR',...
        'Callback','clear all; close all;clc; GUI;');
%GUI es el nombre del programa que se abrira.
```

En la línea 45 se debe reemplazar la palabra *GUI* por el nombre del programa que se busca ejecutar luego de presionar el botón etiquetado como *continuar*. Observar que en este código es posible modificar muchas de las características de la presentación, como el color y tipo de letra, tamaño de la ventana, etc.

Una forma alternativa de presentar el programa (desarrollada por Han Qun) es mostrar una imagen de portada en pantalla durante un determinado tiempo, luego del cual se abrirá la GUI principal. La función mostrada en el Listado 3.2 presenta en pantalla una imagen seleccionada por el usuario durante un tiempo ingresado en milésimas de segundo.

Listado 3.2: Presentación de una imagen en pantalla por un tiempo establecido.

```
1  function presentacion_2(nomb_foto,varargin)
2  % nomb_foto es el nombre de una imagen y su extension.
3  % varargin es el tiempo en milisegundos.
4  % Ejemplo: presentacion_2('presentacion_img.jpg',2000)
5  if nargin ==1
6      I = imread(nomb_foto);
7      time = 4000;
8  elseif (nargin == 2)&&(ischar(varargin{1}))
9      fmt = varargin{1};
10     I = imread(nomb_foto,fmt);
11     time = 4000;
12 elseif (nargin == 2)&&(isnumeric(varargin{1}))
13     I = imread(nomb_foto);
14     time = varargin{1};
15 elseif nargin == 3
16     fmt = varargin{1};
17     I = imread(nomb_foto,fmt);
18     time = varargin{2};
19     if (~isnumeric(time))|| (length(time)~=1)
20         error('ERROR: TIME debe ser un valor numerico en seg.
               ');
21     end
22 else
23     error('ERROR: demasiados datos entrada!');
24 end
25 % Modifica el tamano de la imagen en funcion de la pantalla
26 scrsz = get(0, 'ScreenSize');
27 I=imresize(I,round([scrsz(4)/2 scrsz(3)/2]));
28
29 judasImage = im2java(I);
30 win = javax.swing.JWindow;
31 icon = javax.swing.ImageIcon(judasImage);
32 label = javax.swing.JLabel(icon);
33 win.getContentPane.add(label);
34 screenSize = win.getToolkit.getScreenSize;
35 screenHeight = screenSize.height;
36 screenWidth = screenSize.width;
37 imgHeight = icon.getIconHeight;
38 imgWidth = icon.getIconWidth;
39 win.setLocation((screenWidth-imgWidth)/2,(screenHeight-
     imgHeight)/2);
40 win.pack
```

```
41 win.show
42 tic;
43 while toc < time/1000
44 end
45 win.dispose()
```

Recordar que para usar esta función se debe guardar el archivo .m con el mismo nombre de la función. (Ver Figura 3.2.). Algunas versiones de Matlab informan de este cambio, caso sea necesario.

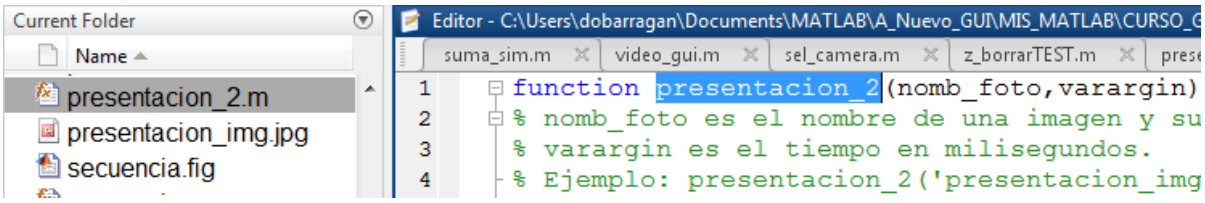

Figura 3.2: El nombre de la función debe ser el mismo del archivo.

Estos programas de presentación son opcionales y dependiendo de la aplicación pueden ser usados o no. A continuación se verá la programación y características de cada elemento de la GUI y la relación entre cada función asociada.

3.2 GUI de suma de dos números.

La forma más sencilla de entender la mecánica de los elementos gráficos de una GUI es a través de una calculadora sencilla (Figura 3.3), debido a que en este programa se sintetiza el funcionamiento de la función de lectura y escritura de datos por el usuario en cada componente de la GUI . En este ejemplo, también se colocará una imagen de fondo que en muchos casos mejora la estética del programa.

Figura 3.3: Entorno del programa sumadora.m.

Conforme se muestra en la Figura 3.3, el usuario ingresa dos valores numéricos en cada uno de los campos de edición de texto que serán sumados al presionar un botón que, asimismo, presenta el resultado en un cuadro de texto estático. El primer paso es ejecutar en la ventana de comandos la función » *guide* (en minúsculas) y a continuación presionar la tecla *enter*. Seleccionamos *Blank GUI (default)* y presionamos *OK*. Insertamos los componentes que muestra la Figura 3.4.

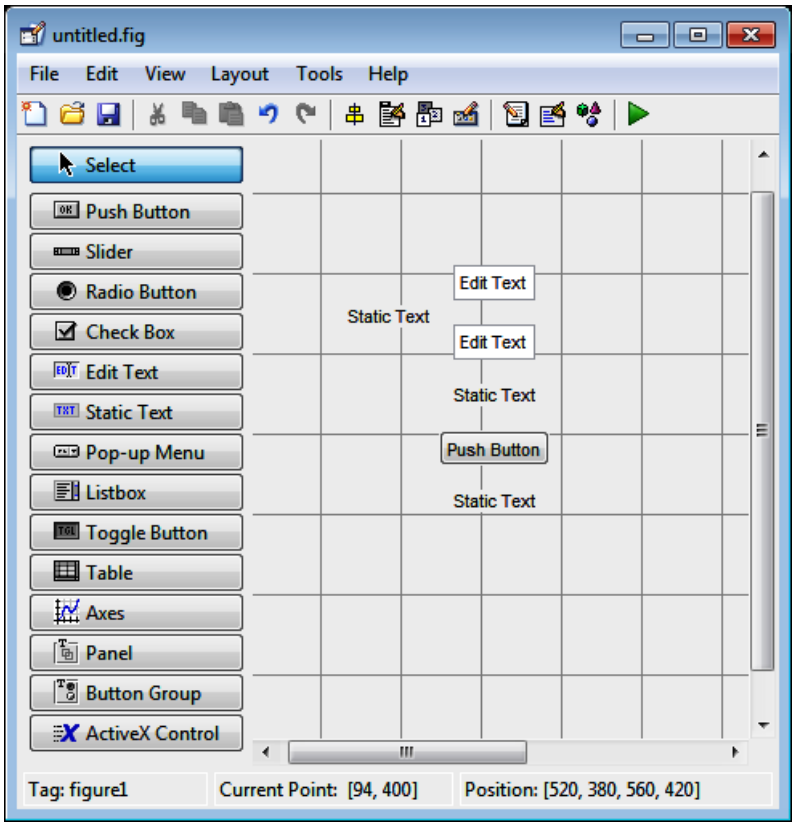

Figura 3.4: Componentes del programa sumadora.

Al hacer doble clic en cada componente en el archivo .fig, se accede a configurar sus propiedades gráficas, como nombre, tamaño, color, etc. Iniciando en el *pushbutton*, configuramos el campo *tag* (etiqueta) y el campo *string*, conforme mostrado en la Figura 3.5.

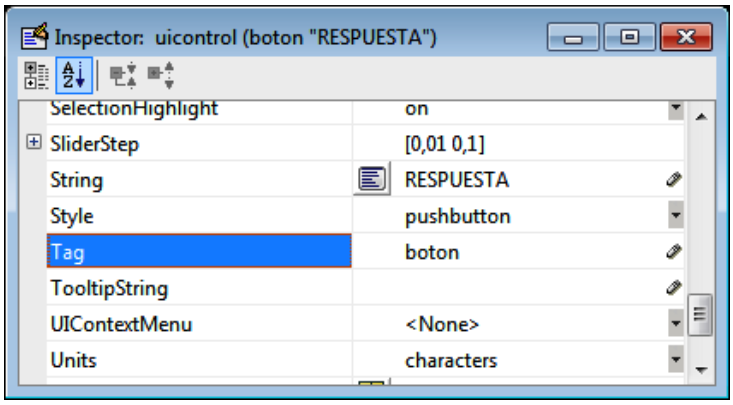

Figura 3.5: Editar propiedades del componente.

El texto en *tag* es el nombre con el que aparecerá la función asociada del *pushbutton* en el archivo .m asociado. El texto en *string* es el nombre con que aparece etiquetado

el botón en el archivo .fig. Es muy recomendable editar nombres apropiados en los campos *tag* de cada componente, ya que esto facilitará la navegación por las funciones asociadas en el archivo .m.

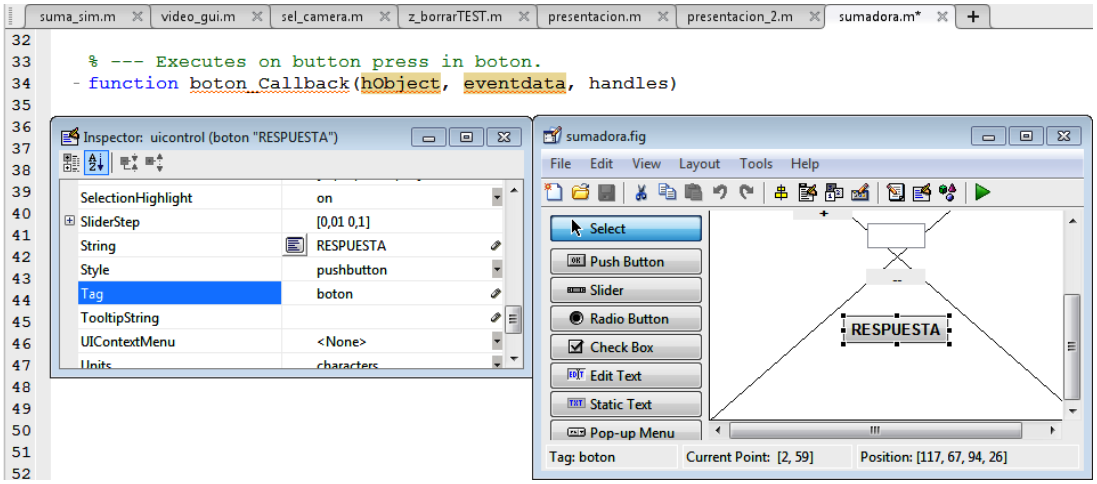

Figura 3.6: Propiedad *tag* y *string* del componente.

Se editan los demás componentes hasta llegar a tener una presentación semejante a la Figura 3.3. De la imagen de fondo se tratará al final de este ejemplo. Una vez colocados todos los elementos gráficos, es tiempo de ejecutar (y guardar) el programa. En el archivo .fig, presionar Ctrl+T para guardar el programa con el nombre *sumadora* en la carpeta MIS_MATLAB (recuerde que Matlab hace diferencia entre mayúsculas y minúsculas. Asimismo, nunca empiece el nombre de un archivo con números ni guiones bajos, no use tildes ni la letra ñ. Para separar dos palabras use guión bajo, ejemplo: mi_programa). En la ventana *Current Directory* aparecen el archivo *sumadora.fig* y *sumadora.m*. (Ver Figura 3.7.)

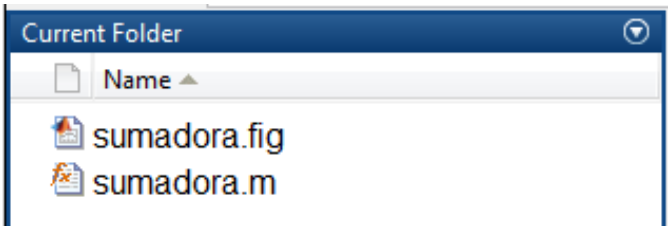

Figura 3.7: Archivos en *Current Directory*.

En el archivo .m se localiza cada función con el botón *Go To*, conforme mostrado en la Figura 3.8. Mientras más funciones existan en la GUI, la ubicación en cada una de ellas resulta sencilla con el uso de este botón. Notar en esta Figura que conviene editar el campo *tag* de cada componente de forma apropiada.

Cada uno de los elementos añadidos en el diseño del archivo .fig como *pushbutton* y *edit text* tienen una función asociada en el archivo .m, cuyo nombre corresponderá al que contenga el campo *tag* del *Property inspector*. Así, al añadir *pushbutton* (tag: boton), tenemos el código del Listado 3.3.

Listado 3.3: Función asociada al pushbutton.

```
1 % --- Executes on button press in boton.
2 function boton_Callback(hObject, eventdata, handles)
```

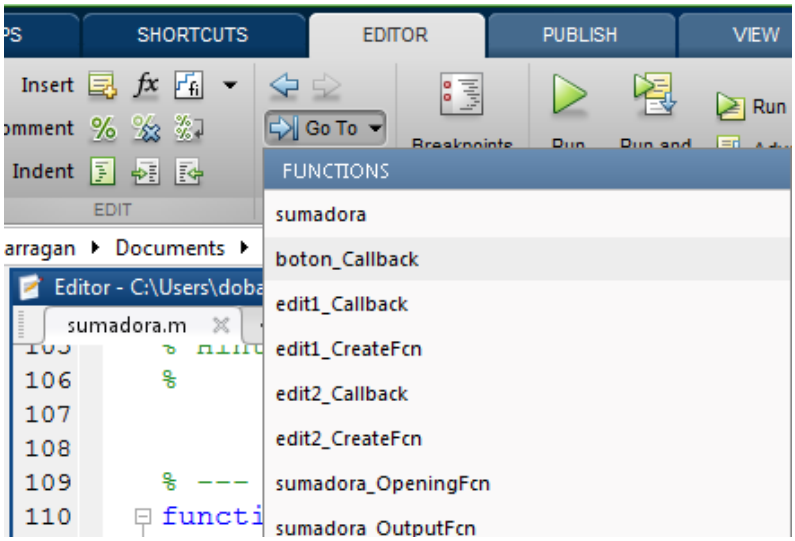

Figura 3.8: Botón *Go To*.

```
3 % hObject    handle to boton (see GCBO)
4 % eventdata  reserved - to be defined in a future version of
    MATLAB
5 % handles    structure with handles and user data (see
    GUIDATA)
```

Cada *edit text* tendrá el código mostrado en el Listado 3.4. Este elemento, como otros, creará de forma automática una función adicional etiquetada como *CreateFcn*. En ella se coloca código de apertura de cada componente. Sin embargo, como se muestra en los tips de GUI al final del manual, esta función puede ser removida sin perjudicar la ejecución correcta del programa. Por el momento, se ignorarán estas funciones.

Listado 3.4: Función asociada al *edit text*.

```
1 function edit1_Callback(hObject, eventdata, handles)
2 % hObject    handle to edit1 (see GCBO)
3 % eventdata  reserved - to be defined in a future version of
    MATLAB
4 % handles    structure with handles and user data (see
    GUIDATA)
5
6 % Hints: get(hObject,'String') returns contents of edit1 as
    text
7 % str2double(get(hObject,'String')) returns contents of edit1
    as a double
```

El elemento *static text* no posee función asociada, pero sí una dirección asociada, que será utilizada para escribir en este componente los resultados de la operación de suma. Para saber cuál es esta dirección, hacer doble clic en este componente y ubicar el campo *tag*, como lo muestra la Figura 3.9.

Hasta ahora se tiene listo el archivo .m para iniciar la programación de la GUI. Notar que los cambios en el campo *tag* de las propiedades de cada elemento modifican el nombre de la función en el archivo .m asociado. De este modo resulta más sencillo navegar por este archivo y editarlo. En la función asociada al botón se edita el código mostrado en el Listado 3.5.

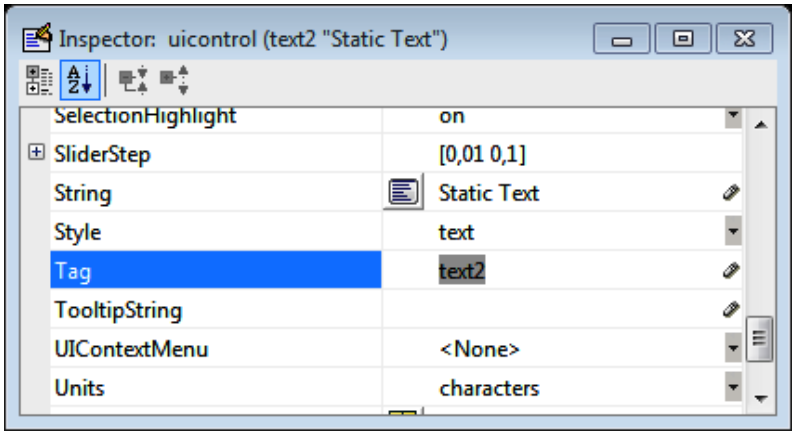

Figura 3.9: Dirección de *static text* (*tag*).

Listado 3.5: Código que se ejecuta al presionar el botón de la GUI.

```
1 function boton_Callback(hObject, eventdata, handles)
2 sumando_1=get(handles.edit1,'String');
3 sumando_2=get(handles.edit2,'String');
4 sumando_1=str2double(sumando_1);
5 sumando_2=str2double(sumando_2);
6 total=sumando_1 + sumando_2;
7 set(handles.text2,'String',total)
```

Notar otra vez que el nombre de la función asociada al botón ha sido etiquetada conforme el nombre asignado en el archivo *.fig*. Esto facilita la negación por el programa al aumentar más funciones. Recordar que los *edit text* se los etiquetaron como *edit1* y *edit2*. Para obtener el parámetro *string* (o el texto contenido en el *edit text*) desde otra función (en este caso la función del botón) se lo hace con la función *get*. La sintaxis de la línea 2 y 3 del Listado 3.5 es muy simple: *get* (obtener) de *handles.edit1*(*edit text*) el campo *string*. Es decir, grabar en una variable lo que el usuario ingresó en la caja de edición de texto.

La variable *sumando_1* y *sumando_2* contiene un valor tipo *string*, el cual se debe modificar de formato con la función *str2double*, con el fin de poder sumar estas cantidades. La suma se realiza con la operación de suma (+) asignando este resultado a la variable *total*. Usando la función *set* se establece (o escribe) el resultado en el *static text* etiquetado como *text2*. La sintaxis de la línea 7 del Listado 3.5 es: *set* (establezca) en *handles.text2* (*static text*) en su campo *string* el valor e la variable *total* (resultado de la suma).

La imagen de fondo puede ser una imagen con extensión *jpg*. Añadir al diseño el componente *axes* y en el campo *tag* del *Property Inspector* editarlo con el nombre *axes_fondo*. El código del Listado 3.6, colocado en la función de apertura (*OpeningFcn*), lee la imagen de fondo y la grafica en el *axes* añadido.

Listado 3.6: Código que se ejecuta al arrancar la GUI: colocación de imagen de fondo.

```
1 function sumadora_OpeningFcn(hObject, eventdata, handles,
     varargin)
2 fondo = imread('img_fondo.jpg');
3 axes(handles.axes_fondo)
```

```
4 axis off
5 image(fondo)
6 handles.output = hObject;
7 guidata(hObject, handles);
```

Notar que se usa el comando *imread* para leer la imagen y *image* para colocarla en *handles.axes_fondo*. En el Listado 3.7 se encuentra el código completo de esta GUI. Se han eliminado los comentarios que Matlab genera de forma automática para cada función asociada.

Listado 3.7: Código del programa sumadora.

```
1  function varargout = sumadora(varargin)
2  % Begin initialization code - DO NOT EDIT
3  gui_Singleton = 1;
4  gui_State = struct('gui_Name',       mfilename, ...
5                     'gui_Singleton',  gui_Singleton, ...
6                     'gui_OpeningFcn', @sumadora_OpeningFcn,
                          ...
7                     'gui_OutputFcn',  @sumadora_OutputFcn, ...
8                     'gui_LayoutFcn',  [] , ...
9                     'gui_Callback',   []);
10 if nargin && ischar(varargin{1})
11     gui_State.gui_Callback = str2func(varargin{1});
12 end
13
14 if nargout
15     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
           {:});
16 else
17     gui_mainfcn(gui_State, varargin{:});
18 end
19 % End initialization code - DO NOT EDIT
20 % --- Executes just before sumadora is made visible.
21 function sumadora_OpeningFcn(hObject, eventdata, handles,
       varargin)
22 fondo = imread('img_fondo.jpg');
23 axes(handles.axes_fondo)
24 axis off
25 image(fondo)
26 handles.output = hObject;
27 guidata(hObject, handles);
28
29 % --- Outputs from this function are returned to the command
       line.
30 function varargout = sumadora_OutputFcn(hObject, eventdata,
       handles)
31 varargout{1} = handles.output;
32
33 % --- Executes on button press in boton.
34 function boton_Callback(hObject, eventdata, handles)
35 sumando_1=get(handles.edit1,'String');
36 sumando_2=get(handles.edit2,'String');
37 sumando_1=str2double(sumando_1);
38 sumando_2=str2double(sumando_2);
```

```
39 total=sumando_1 + sumando_2;
40 set(handles.text2,'String',total)
41
42 function edit1_Callback(hObject, eventdata, handles)
43
44 function edit1_CreateFcn(hObject, eventdata, handles)
45 if ispc && isequal(get(hObject,'BackgroundColor'),...
46         get(0,'defaultUicontrolBackgroundColor'))
47     set(hObject,'BackgroundColor','white');
48 end
49
50 function edit2_Callback(hObject, eventdata, handles)
51
52 function edit2_CreateFcn(hObject, eventdata, handles)
53 if ispc && isequal(get(hObject,'BackgroundColor'),...
54         get(0,'defaultUicontrolBackgroundColor'))
55     set(hObject,'BackgroundColor','white');
56 end
```

Es posible programar el código del botón de suma reduciendo el número de líneas. Esto se muestra en el Listado 3.8. No obstante en caso el programa vaya a ser analizado por otros programadores, esto podría reducir la claridad del algoritmo.

Listado 3.8: Código del botón de suma.

```
1 function boton_Callback(hObject, eventdata, handles)
2 ttl=str2double(get(handles.edit1,'String'))+str2double(get(
     handles.edit2,'String'));
3 set(handles.text2,'String',ttl)
```

Cada función asociada que se genera de forma automática al añadir un nuevo componente en la ventana .fig viene acompañada de un conjunto de comentarios explicativos. No obstante, para evitar que Matlab coloque comentarios automáticos en cada nueva función generada, se modifica la última opción en las preferencias de GUI. (Ver Figura 3.10).

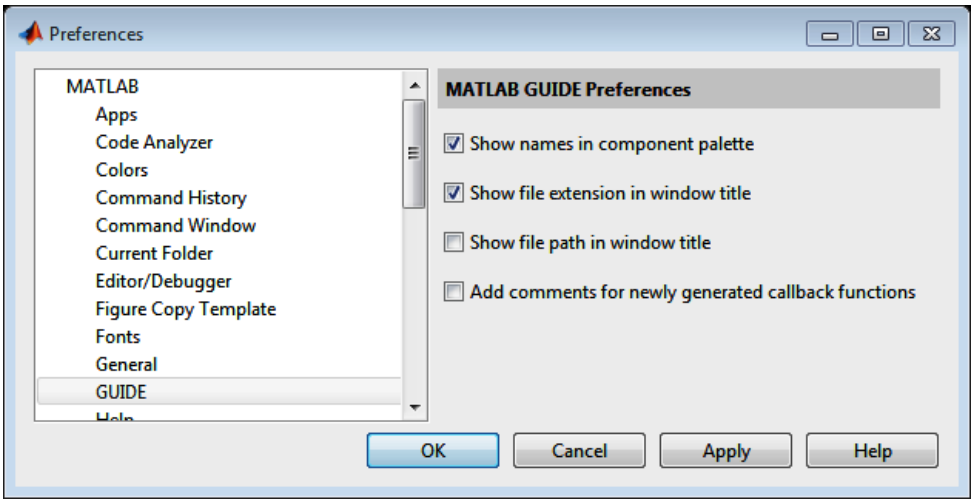

Figura 3.10: Preferencias de la GUI.

3.3 Sumadora automática.

En este programa, indicado en la Figura 3.11, se mostrará la ejecución de cálculos con el simple hecho de ingresar datos en un *edit text*, es decir, sin necesidad de un *pushbutton*.

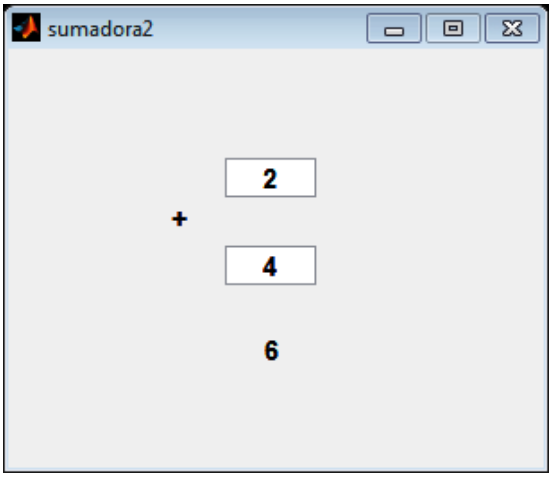

Figura 3.11: GUI sumadora con cálculo automático.

Lo primero a realizar es modificar el campo *tag* de cada elemento conforme mostrado en la Tabla 3.1. Notar que los nombres asignados a cada campo *tag* son relacionados con la función de cada componente.

Tabla 3.1: Campo *Tag* de cada elemento de la GUI.

Elemento	Tag
Edit text 1	uno_edit
Edit text 2	dos_edit
Static text	respuesta

Con un clic derecho en el primer *edit text* se selecciona *View Callbacks, Callback*. Esto ubicará el cursor de pantalla en la parte del archivo .m correspondiente para este *edit text*. Modificamos el código de esta función conforme se muestra en el Listado 3.9.

Listado 3.9: Código del *edit text*.

```
function uno_edit_Callback(hObject, eventdata, handles)
sum1=str2double(get(hObject,'String'));
sum2=str2double(get(handles.dos_edit,'String'));
if isnan(sum1)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.uno_edit,'String',0);
    sum1=0;
end
if isnan(sum2)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.dos_edit,'String',0);
    sum2=0;
end
```

```
14 suma=sum1+sum2;
15 set(handles.respuesta,'String',suma);
16 guidata(hObject, handles);
```

Como se puede ver, es posible añadir la detección y corrección de errores de los datos ingresados en cada *edit text* usando una función de verificación del formato del dato ingresado. El resultado de la función *isnan(sum1)* ejcutado en la ventana de comandos se muestra en el Listado 3.10.

Listado 3.10: isnan para verificar dato ingresado.

```
1 >> isnan(str2double('a'))
2 ans =
3 1
4 >> isnan(str2double('0'))
5 ans =
6 0
```

De tal manera, la sentencia *if isnan(sum1)*, en caso de dar un 1, presentará un cuadro de error (estos mensajes de usuario se ven más adelante) y corrige el valor asignando 0 al sumando donde se ha ingresado el error con la función *set(handles.dos_edit,'String',0); sum2=0*.

El código correspondiente al segundo elemento de edición de texto se muestra en el Listado 3.11.

Listado 3.11: Código del segundo *edit text*

```
1  function dos_edit_Callback(hObject, eventdata, handles)
2  sum1=str2double(get(hObject,'String'));
3  sum2=str2double(get(handles.uno_edit,'String'));
4  if isnan(sum1)
5      errordlg('El valor debe ser numerico','ERROR')
6      set(handles.dos_edit,'String',0);
7      sum1=0;
8  end
9  if isnan(sum2)
10     errordlg('El valor debe ser numerico','ERROR')
11     set(handles.uno_edit,'String',0);
12     sum2=0;
13 end
14 suma=sum1+sum2;
15 set(handles.respuesta,'String',suma);
16 guidata(hObject, handles);
```

Para eliminar las funciones *CreateFcn* que son generadas de forma automática por Matlab son necesarios dos cambios:

- Eliminar de forma manual en el editor el código generado en el archivo .m: buscar la función *function uno_edit_CreateFcn(hObject, eventdata, handles)*. Eliminar esta función y la asociada con *dos_edit*.
- Eliminar la referencia a la función en el *property inspector* del archivo .fig: conforme mostrado en la Figura 3.12.

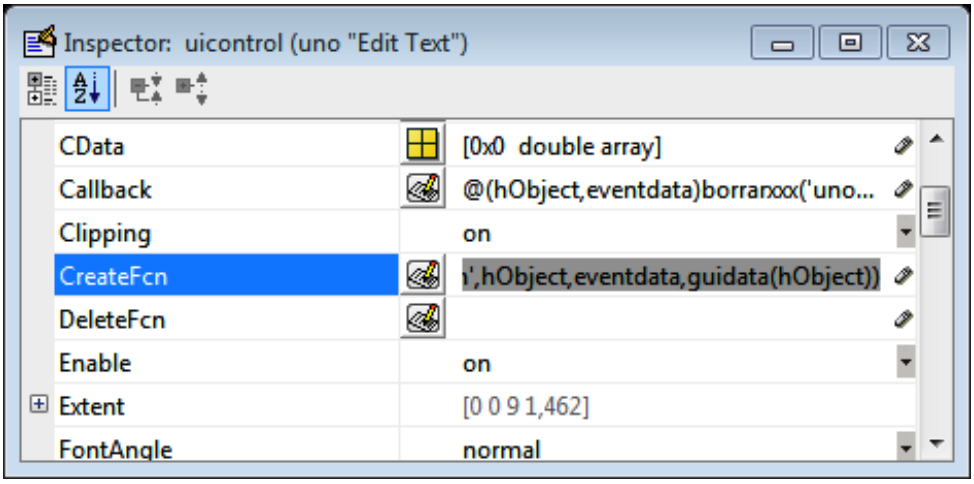

Figura 3.12: GUI sumadora con cálculo automático.

El código completo correspondiente a esta GUI se presenta en el Listado 3.12.

Listado 3.12: Código sumadora 2.

```
function varargout = sumadora2(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @sumadora2_OpeningFcn, ...
                   'gui_OutputFcn',  @sumadora2_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end
if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before sumadora2 is made visible.
function sumadora2_OpeningFcn(hObject, eventdata, handles, varargin)
handles.output = hObject;
guidata(hObject, handles);
% --- Outputs from this function are returned to the command line.
function varargout = sumadora2_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
%
function uno_edit_Callback(hObject, eventdata, handles)
sum1=str2double(get(hObject,'String'));
sum2=str2double(get(handles.dos_edit,'String'));
```

```
if isnan(sum1)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.uno_edit,'String',0);
    sum1=0;
end
if isnan(sum2)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.dos_edit,'String',0);
    sum2=0;
end
suma=sum1+sum2;
set(handles.respuesta,'String',suma);
guidata(hObject, handles);
%
function dos_edit_Callback(hObject, eventdata, handles)
sum1=str2double(get(hObject,'String'));
sum2=str2double(get(handles.uno_edit,'String'));
if isnan(sum1)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.dos_edit,'String',0);
    sum1=0;
end
if isnan(sum2)
    errordlg('El valor debe ser numerico','ERROR')
    set(handles.uno_edit,'String',0);
    sum2=0;
end
suma=sum1+sum2;
set(handles.respuesta,'String',suma);
guidata(hObject, handles);
```

3.4 Uso de *align objects*: calculadora.

Con este ejemplo exploraremos las funciones *strcat* (*string concatenation*) y *eval*. Asimismo, usaremos un cuadro mensaje de diálogo y la herramienta *Align Objects*.

Abrir una nueva GUI y colocar cada uno de los componentes de la Figura 3.13. Luego, añadir un panel y arrastrar los demás elementos dentro de este componente: diecisiete *pushbutton* y un *static text*. Asimismo, es necesario para este ejemplo usar la herramienta de alineación de objetos. La Figura 3.14 muestra el entrono de *Align Objects*.

El comando *strcat* une o concatena varios *strings*. El Listado 3.13 presenta el modo de uso de esta función.

Listado 3.13: Función *strcat*.

```
>>d='Stieg';
e='_';
b='Larsson';
strcat(d,e,b)
ans =
Stieg_Larsson
```

La función *eval* ejecuta una operación que está dada en formato *string*. El Listado 3.14 presenta el modo de uso de esta función.

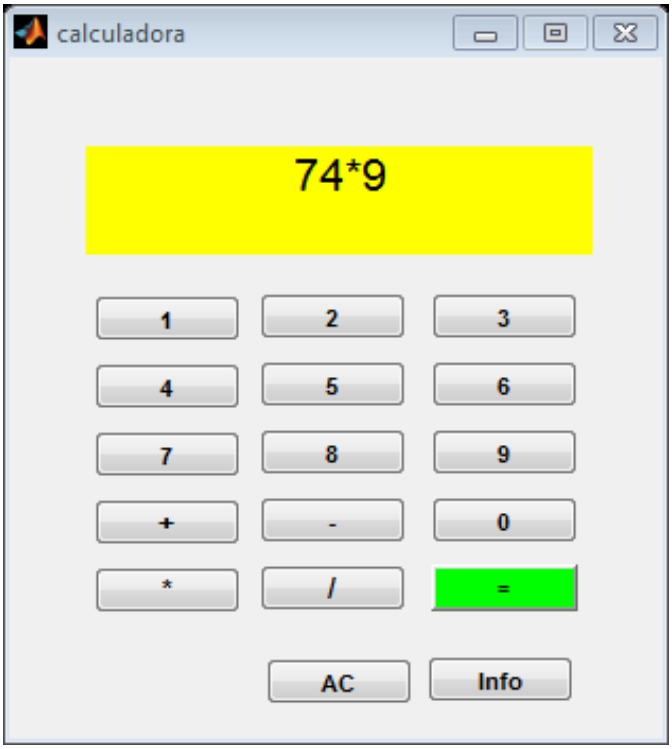

Figura 3.13: GUI calculadora básica.

Listado 3.14: Función *eval*.

```
>>eval('2+3+4')
ans =
9
>>eval('sqrt(144)')
ans =
12
```

Antes de continuar, se hará un breve repaso del *pushbutton*. En un GUI en blanco, añadir un par de *pushbutton*. Grabar las GUI de ejemplo con el nombre *eliminar*. Con clic derecho, ir a la opción que muestra la Figura 3.15.

Inmediatamente, esto nos irá a ubicar en la parte del archivo .m correspondiente a la función resaltada del *pushbutton*.

Listado 3.15: Función asociada al *pushbutton*.

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

En *Property Editor*, etiquetar el nombre con el que aparecerá el *pushbutton* en el archivo .m. En el campo *tag*, editar el nombre *uno*, para el primer *pushbutton* y *dos* para el segundo *pushbutton*. Nuevamente, ubicarse en *View Callbacks*,*Callbacks* y notar el cambio de etiqueta en el archivo .m.

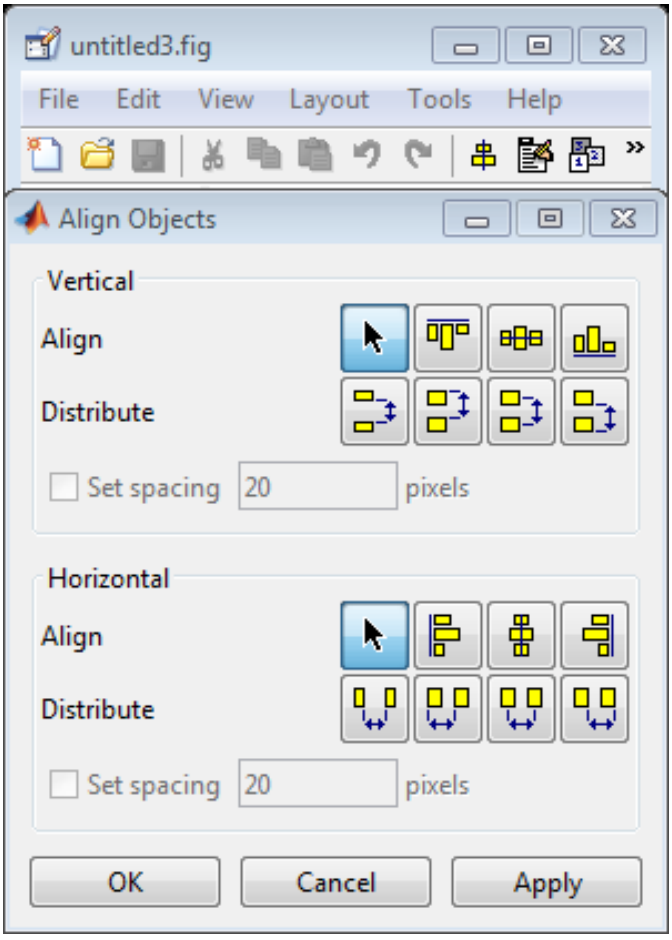

Figura 3.14: Herramienta de alineación de componente de la GUI.

Listado 3.16: Función asociada al *pushbutton* con el campo *tag* editado.

```
% --- Executes on button press in uno.
function uno_Callback(hObject, eventdata, handles)
% hObject    handle to uno (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Gracias a esta forma de etiquetación, en un diseño que contenga muchos botones (como el caso de la calculadora), es sencillo una mejor ubicación en la programación del archivo .m. Al ejecutar *Show Functions*, se tendrá la presentación de la Figura 3.16 donde se muestra la etiqueta de cada botón de la GUI.

Empezando en *uno_Callback*, escribimos el código del Listado 3.17.

Listado 3.17: Función asociada al *pushbutton* del número 1.

```
function uno_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=strcat(dato,'1');
set(handles.text1,'String',dato)
```

Repetir el mismo código para los demás botones, excepto para el botón igual, donde se coloca el código del Listado 3.18.

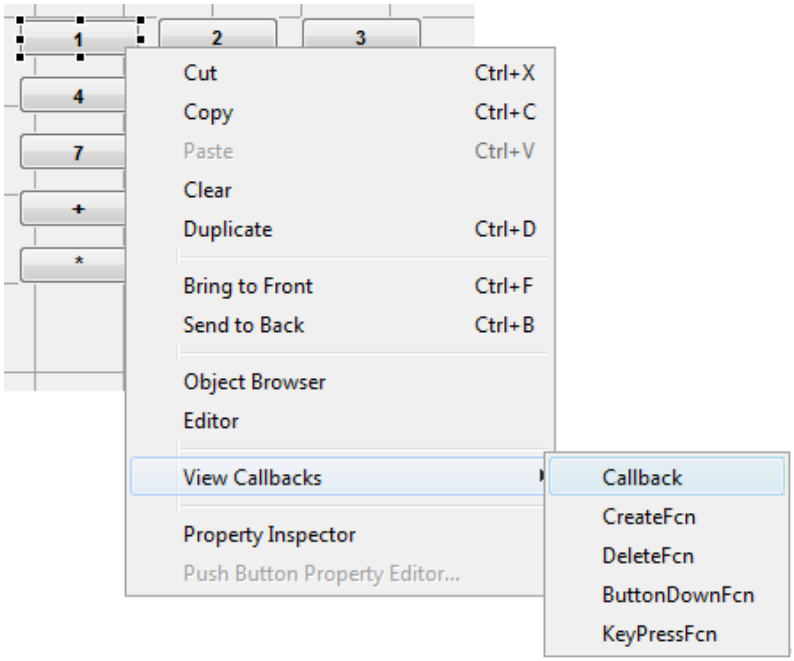

Figura 3.15: Ubicación de la función asociada al *push button*.

Listado 3.18: Función asociada al *pushbutton* del cálculo de la respuesta.

```
function igual_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=eval(dato);
set(handles.text1,'String',dato)
```

El botón de borrado, etiquetado como *borrar*, se configura según las funciones del Listado 3.19.

Listado 3.19: Función asociada al *pushbutton* de borrado.

```
function borrar_Callback(hObject, eventdata, handles)
set(handles.text1,'String',' ')
```

La función asociada al cuadro de diálogo mostrará un mensaje de información. El Listado 3.19 presenta esta programación.

Listado 3.20: Función asociada al *pushbutton* del cuadro de diálogo.

```
function info_Callback(hObject, eventdata, handles)
msgbox('Calculadora Sencilla','Acerca de...');
```

Esto sería todo en la calculadora sencilla. Notar que es posible crear alguno botones adicionales, como el punto decimal, raíz cuadrada, etc. El Listado 3.21 contiene el código completo de este programa.

Listado 3.21: Código del programa calculadora.

```
function varargout = calculadora(varargin)
% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
```

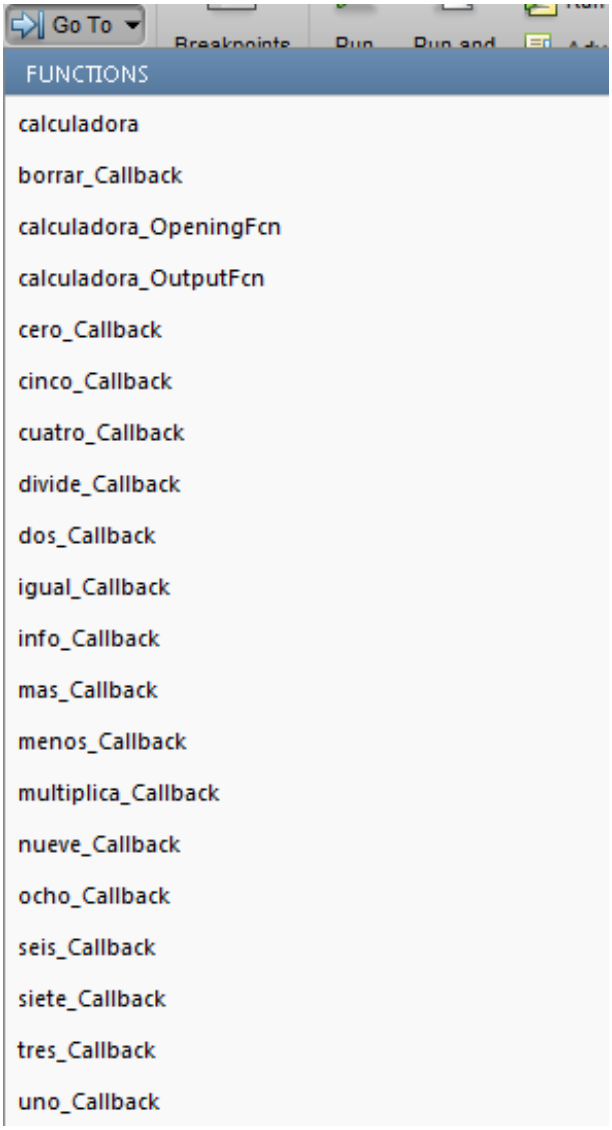

Figura 3.16: Listado de las funciones asociadas a cada componente de la GUI.

```
5                    'gui_Singleton',  gui_Singleton, ...
6                    'gui_OpeningFcn', @calculadora_OpeningFcn,
                        ...
7                    'gui_OutputFcn',  @calculadora_OutputFcn,
                        ...
8                    'gui_LayoutFcn',  [] , ...
9                    'gui_Callback',   []);
10 if nargin && ischar(varargin{1})
11     gui_State.gui_Callback = str2func(varargin{1});
12 end
13
14 if nargout
15     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin
           {:});
16 else
17     gui_mainfcn(gui_State, varargin{:});
18 end
19 % End initialization code - DO NOT EDIT
20 % --- Executes just before calculadora is made visible.
21 function calculadora_OpeningFcn(hObject, eventdata, handles,
```

```
    varargin)
22 handles.output = hObject;
23 guidata(hObject, handles);
24 % --- Outputs from this function are returned to the command
    line.
25 function varargout = calculadora_OutputFcn(hObject, eventdata
    , handles)
26 varargout{1} = handles.output;
27 % --- Executes on button press in uno.
28 function uno_Callback(hObject, eventdata, handles)
29 dato=get(handles.text1,'String');
30 dato=strcat(dato,'1');
31 set(handles.text1,'String',dato)
32 % --- Executes on button press in cuatro.
33 function cuatro_Callback(hObject, eventdata, handles)
34 dato=get(handles.text1,'String');
35 dato=strcat(dato,'4');
36 set(handles.text1,'String',dato)
37 % --- Executes on button press in siete.
38 function siete_Callback(hObject, eventdata, handles)
39 dato=get(handles.text1,'String');
40 dato=strcat(dato,'7');
41 set(handles.text1,'String',dato)
42 % --- Executes on button press in mas.
43 function mas_Callback(hObject, eventdata, handles)
44 dato=get(handles.text1,'String');
45 dato=strcat(dato,'+');
46 set(handles.text1,'String',dato)
47 % --- Executes on button press in multiplica.
48 function multiplica_Callback(hObject, eventdata, handles)
49 dato=get(handles.text1,'String');
50 dato=strcat(dato,'*');
51 set(handles.text1,'String',dato)
52 % --- Executes on button press in dos.
53 function dos_Callback(hObject, eventdata, handles)
54 dato=get(handles.text1,'String');
55 dato=strcat(dato,'2');
56 set(handles.text1,'String',dato)
57 % --- Executes on button press in cinco.
58 function cinco_Callback(hObject, eventdata, handles)
59 dato=get(handles.text1,'String');
60 dato=strcat(dato,'5');
61 set(handles.text1,'String',dato)
62 % --- Executes on button press in ocho.
63 function ocho_Callback(hObject, eventdata, handles)
64 dato=get(handles.text1,'String');
65 dato=strcat(dato,'8');
66 set(handles.text1,'String',dato)
67 % --- Executes on button press in menos.
68 function menos_Callback(hObject, eventdata, handles)
69 dato=get(handles.text1,'String');
70 dato=strcat(dato,'-');
71 set(handles.text1,'String',dato)
72 % --- Executes on button press in divide.
73 function divide_Callback(hObject, eventdata, handles)
```

```
dato=get(handles.text1,'String');
dato=strcat(dato,'/');
set(handles.text1,'String',dato)
% --- Executes on button press in tres.
function tres_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=strcat(dato,'3');
set(handles.text1,'String',dato)
% --- Executes on button press in seis.
function seis_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=strcat(dato,'6');
set(handles.text1,'String',dato)
% --- Executes on button press in nueve.
function nueve_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=strcat(dato,'9');
set(handles.text1,'String',dato)
% --- Executes on button press in cero.
function cero_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=strcat(dato,'0');
set(handles.text1,'String',dato)
% --- Executes on button press in igual.
function igual_Callback(hObject, eventdata, handles)
dato=get(handles.text1,'String');
dato=eval(dato);
set(handles.text1,'String',dato)
% --- Executes on button press in borrar.
function borrar_Callback(hObject, eventdata, handles)
set(handles.text1,'String',' ')
% --- Executes on button press in info.
function info_Callback(hObject, eventdata, handles)
msgbox('Calculadora Sencilla','Acerca de...');
```

3.5 Mensajes de usuario.

Como se vio en el ejemplo de la *sumadora2* y la *calculadora*, es posible añadir un cuadro de mensaje para el usuario. Existen algunos tipos de cuadros de mensaje. Como ejemplo, se creará un nueva GUI con el nombre *mensajes*. Luego de añadir un panel y dentro de él cinco *pushbutton*, en el *property inspector* se edita los nombres como muestra la Figura 3.17. Editamos el campo *tag* con los mismos nombres.

Las funciones del Listado 3.22 se ubican debajo de la función correspondiente.

Listado 3.22: Funciones que generan mensajes de usuario.

```
warndlg('Esto es un aviso','Curso_GUIDE');
errordlg('Esto es un mensaje de error',' Curso_GUIDE ');
helpdlg('Esto es una ayuda',' Curso_GUIDE ');
msgbox('Esto es un cuadro de mensaje',' Curso_GUIDE ');
questdlg('Esto es una pregunta',' Curso_GUIDE ');
inputdlg('DATO','Encabezado');
```

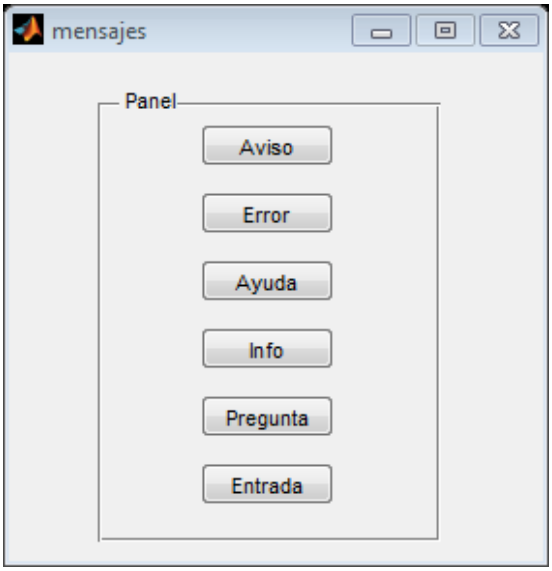

Figura 3.17: GUI de prueba de mensajes de usuario.

Parte del archivo .m queda conforme muestra el Listado 3.23.

Listado 3.23: Funciones para mensajes de usuario en el m-file.

```
% --- Executes on button press in aviso.
function aviso_Callback(hObject, eventdata, handles)
warndlg('Esto es un aviso','Curso_GUIDE');
% --- Executes on button press in info.
function info_Callback(hObject, eventdata, handles)
msgbox('Esto es un cuadro de mensaje',' Curso_GUIDE ');
% --- Executes on button press in error.
function error_Callback(hObject, eventdata, handles)
errordlg('Esto es un mensaje de error',' Curso_GUIDE ');
% --- Executes on button press in ayuda.
function ayuda_Callback(hObject, eventdata, handles)
helpdlg('Esto es una ayuda',' Curso_GUIDE ');
% --- Executes on button press in pregunta.
function pregunta_Callback(hObject, eventdata, handles)
questdlg('Esto es una pregunta',' Curso_GUIDE ');
% --- Executes on button press in entrada.
function entrada_Callback(hObject, eventdata, handles)
 dato=inputdlg('DATO','Encabezado');
 disp(dato{1})
```

Para el caso especial de las preguntas podemos ejecutar funciones dependiendo de la respuesta escogida. Por ejemplo, si deseamos salir o no del programa usamos el código de Listado 3.24.

Listado 3.24: Programación del mensaje de pregunta.

```
opc=questdlg('Desea salir del programa?','SALIR','Si','No','
   No');
if strcmp(opc,'No')
return;
end
clear,clc,close all
```

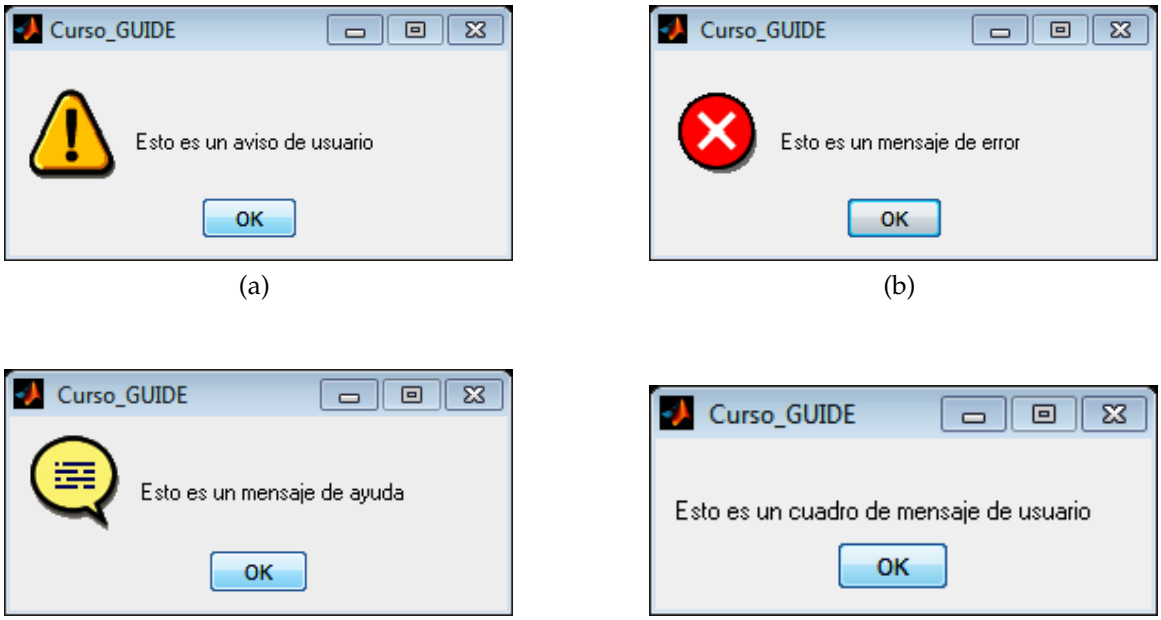

Figura 3.18: (a) Mensaje de alerta. (b) Mensaje de error. (c) Mensaje de ayuda. (d) Mensaje general.

La función *strcmp* compara dos *strings* y si son iguales retorna el valor 1 (*true*). *Clear* elimina todas los valores de *workspace*, *clc* limpia la pantalla y *close all* cierra todos las interfaces gráficas. Nóte que la secuencia 'Si', 'No', 'No' termina en 'No'; con esto se logra que la parte No del cuadro de pregunta esté resaltado. Si terminara en 'Si', la parte Si del cuadro de pregunta se resaltaría.

Esta pregunta es útil para el caso de tener una solicitud de salida del programa, por ejemplo, si al hacer clic derecho en el archivo .fig y seleccionamos lo que muestra la Figura 3.19, podemos editar el código como en el Listado 3.25 para preguntar si se desea salir o no del programa.

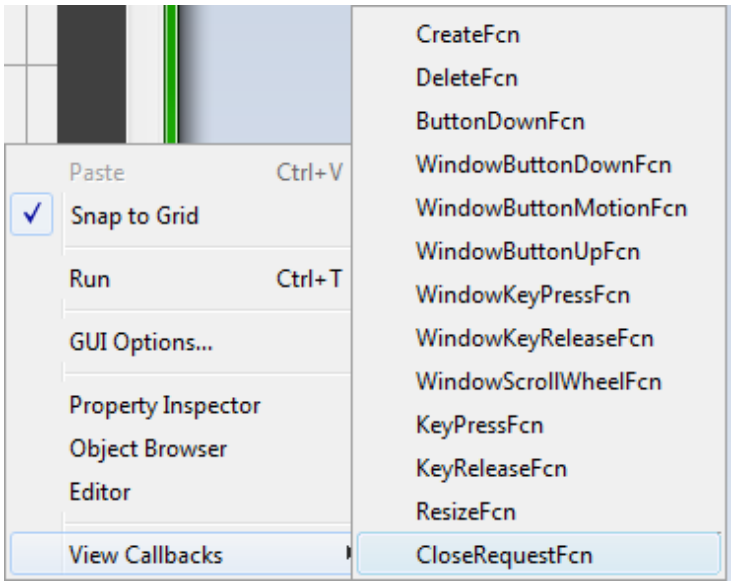

Figura 3.19: Función de solicitud de cierre.

En la función *CloseRequestFcn* del Listado 3.25 la función *questdlg* presentará una ventana con una pregunta y botones con opciones de salir o no del programa.

Listado 3.25: Configuración de la opción de pregunta en la función *Close Request*.

```
function figure1_CloseRequestFcn(hObject, eventdata, handles)
opc=questdlg('Desea salir del programa',...
'SALIR','Si','No','No');
if strcmp(opc,'No')
return;
end
delete(hObject);
%
```

Estos mensajes de usuario poseen una sola línea de información. La función *errordlg('Hola','Un', 'Saludo','Mensaje de error')* agregar saltos de línea.

Otra función interesante es *uigetfile*, que permite abrir un archivo y obtener su nombre y dirección. Al ejecutar el Listado 3.26 en el *command window* se mostrará la pantalla de selección de archivo. (Ver Figura 3.20.)

Listado 3.26: Retorno de la función *uigetfile*.

```
[FileName Path]=uigetfile({'*.m;*.mdl'}, 'Escoger')
```

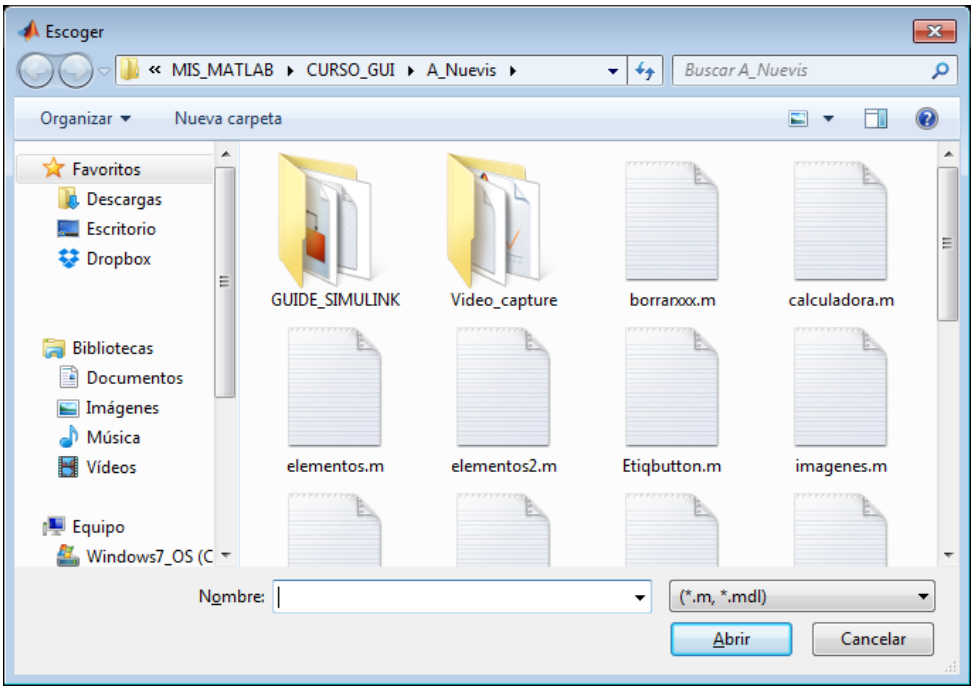

Figura 3.20: GUI de selección de archivo.

Al escoger un programa cualquiera, esta función retorna:

Listado 3.27: Retorno de la función *uigetfile*.

```
FileName =
qpsk_mod_const_freq.mdl
Path =
C:\MATLAB71\work\
```

Listado 3.28: Retorno de la función *uigetfile* cuando se presiona Cancelar.

```
FileName =
0
Path =
0
```

Con esta información podemos abrir cualquier archivo o ejecutar cualquier programa. Por ejemplo, cargar una imagen en un *axes*, abrir un documento *excel* o correr un software específico. Creemos una GUI con tres *pushbuttons* (el campo *tag* es igual al campo *string*) y un *axes* como muestra la Figura 3.21.

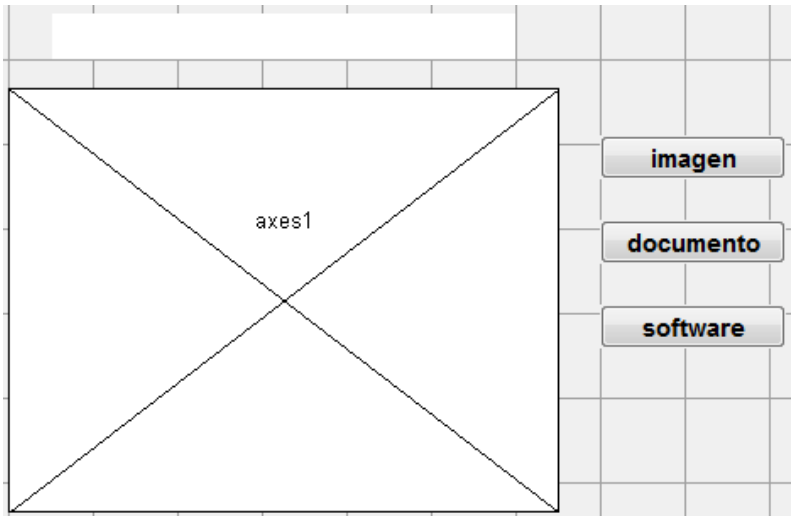

Figura 3.21: Uso de *uigetfile* en una GUI.

Al ubicarse en la función del botón imagen, editamos el código del Listado 3.29. En este caso, usamos apenas dos formatos de tipo de imagen, mas esta función puede incluir el resto de formatos como *png* o *eps* por ejemplo. La función *uigetfile* permite también añadir un mensaje de etiquetación a la ventana de selección.

Listado 3.29: Abrir imagen con *uigetfile*.

```
function imagen_Callback(hObject, eventdata, handles)
[FileName,Path]=uigetfile({'*.jpg;*.bmp'},'Abrir imagen');
if isequal(FileName,0)
    return
else
    a=imread(strcat(Path,FileName));
    imshow(a);
end
handles.direccion=strcat(Path,FileName);
set(handles.text1,'String',FileName)
guidata(hObject,handles)
```

Como se puede observar en el código listado, si el usuario presiona cancelar, el programa no ejecuta ningún proceso. La función *imread* lee una imagen en Matlab como una matiz y la función *imshow* la presenta en el axes correspondiente (se puede también la función *image* para presentar la imagen). (Ver Figura 3.22.)

El programa del botón *documento* se indica en el Listado 3.30.

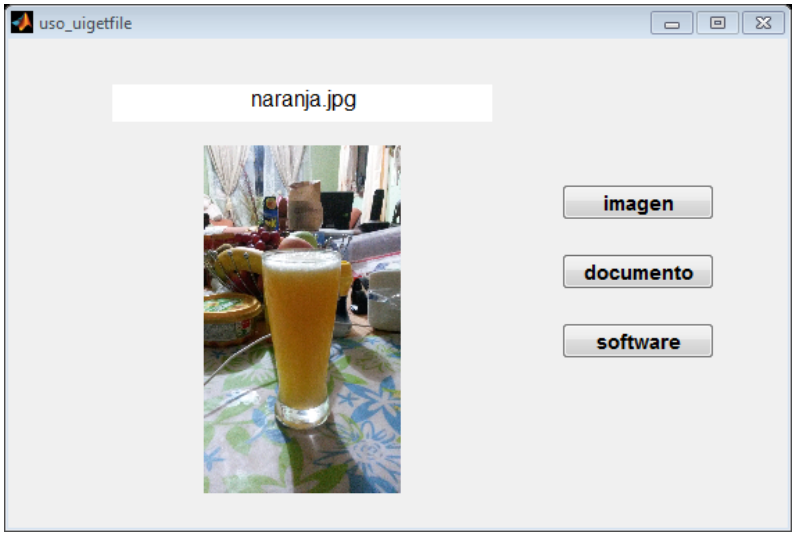

Figura 3.22: Uso de *uigetfile* para abrir una imagen.

Listado 3.30: Abrir un documento con *uigetfile*.

```
function documento_Callback(hObject, eventdata, handles)
[FileName,Path]=uigetfile({'*.doc;*.xls;*.docx;*.xlsx'},'
   Abrir documento');
if isequal(FileName,0)
    return
else
    winopen(strcat(Path,FileName));
end
```

La función *winopen* abre el documento en la dirección especificada por el *path* y el *filename* del archivo.

El programa del botón programa se muestra en el Listado 3.31.

Listado 3.31: Abrir un documento con *uigetfile*.

```
function software_Callback(hObject, eventdata, handles)
[FileName,Path]=uigetfile({'*.exe'},'Abrir software');
if isequal(FileName,0)
    return
else
    winopen(strcat(Path,FileName));
end
```

3.6 Gráficas de senoides.

Con este ejemplo se mostrará el uso del componente *axes* y del *pop up menu* en una GUI que grafica la función seno, coseno y la suma del ambas funciones. (Ver Figura 3.23.) Creamos un nuevo GUI con el nombre *senoides*. Añadimos al mismo un componente *axes* y un *pop up menu*. Editamos el *string* del *pop up menu* con seno, coseno y suma.

Toda GUI posee una función donde es posible programar las condiciones iniciales que tendrá nuestra GUI (función *OpeningFcn*). En nuestro ejemplo, está etiquetada como *senoides_OpeningFcn*. Debajo de esta línea, editamos el código como muestra el Listado

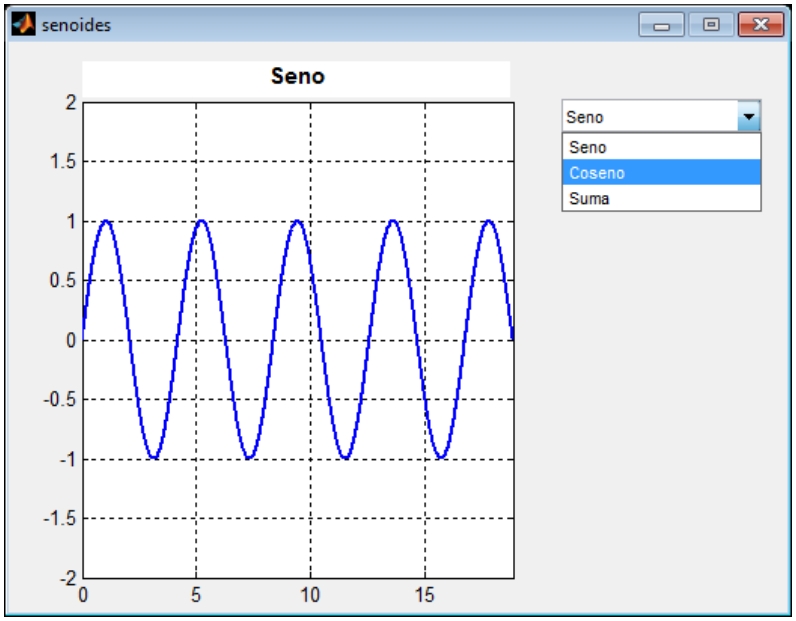

Figura 3.23: Gráfica de una senoide en GUI.

3.32.

Listado 3.32: Función de apertura de la GUI.

```
function senoides_OpeningFcn(hObject, eventdata, handles,
   varargin)
t=0:pi/500:6*pi;
y=sin(1.5*t);
y1=cos(t);
y2=y+y1;
handles.tiempo=t;
handles.seno=y;
handles.coseno=y1;
handles.suma=y2;
handles.current_data=handles.seno;
plot(t,handles.current_data,'LineWidth',2);
grid on
axis([0,6*pi,-2,2])
handles.output = hObject;
guidata(hObject, handles);
```

Con un clic derecho en *pop-up menú*, nos ubicamos en *View Callbacks* y luego en *Callbacks*. Esto nos lleva a la función function del Listado 3.33, en la cual se deberá añadir el código mostrado.

Listado 3.33: Función del pop up menu de la GUI.

```
function popupmenu1_Callback(hObject, eventdata, handles)
t=handles.tiempo;
val=get(hObject,'Value');
str=get(hObject,'String');
if (val==1)
    handles.dato=handles.seno;
elseif (val==2)
    handles.dato=handles.coseno;
```

```
9  else
10     handles.dato=handles.suma;
11 end
12 plot(t,handles.dato,'LineWidth',2);
13 set(handles.text1,'String',str{val})
14 grid on
15 axis([0,6*pi,-2,2])
16 guidata(hObject,handles);
```

3.7 Uso del *listbox*.

Cos este ejemplo, se mostrará el uso de un *listbox*. Creamos un nuevo GUI etiquetado como *Listbox*. Añadimos al diseño un *listbox* y un par de *statictext* y los ordenamos como lo muestra la Figura 3.24.

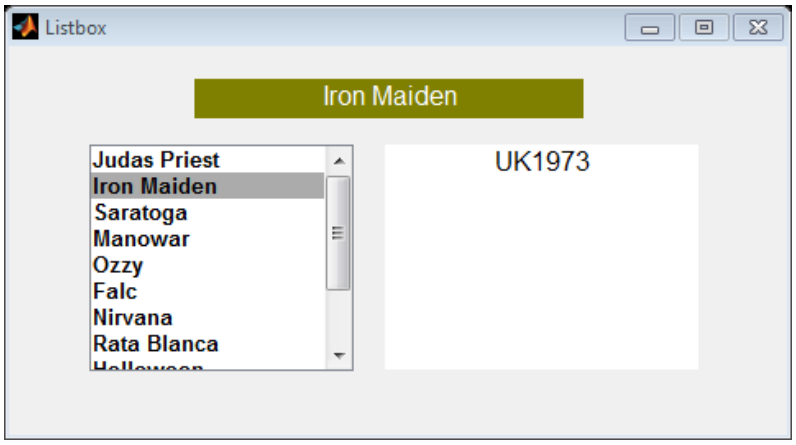

Figura 3.24: Uso de *listbox*.

Haciendo doble clic en *listbox* editamos los elementos de la lista.

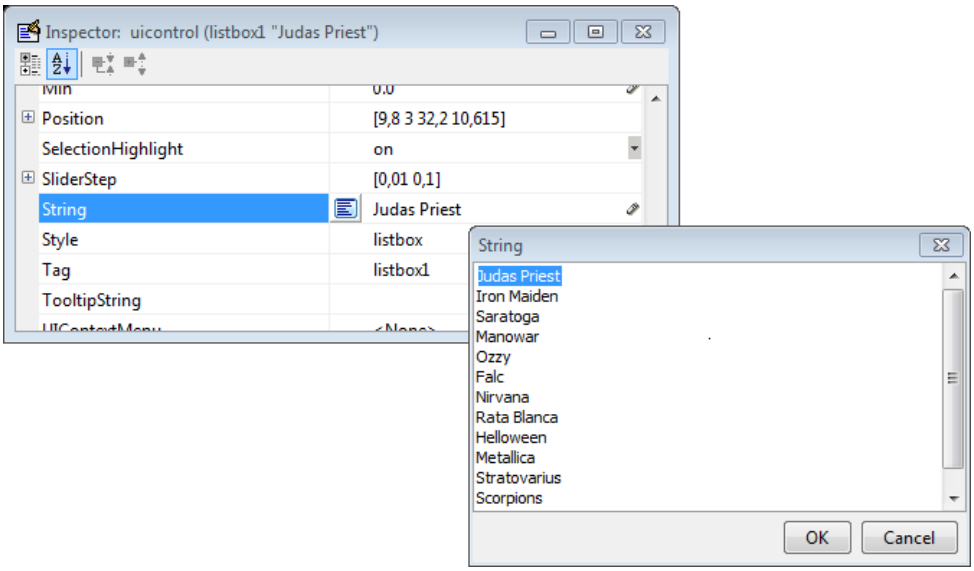

Figura 3.25: Opciones del *listbox*.

Con un click derecho en *listbox*, nos ubicamos en *View Callbacks* y luego en *Callbacks*. Esto nos lleva a la m-file donde se encuentra la función del *listbox*. Editamos el código

del listado 3.34.

Listado 3.34: Función del list box de la GUI.

```
function listbox1_Callback(hObject, eventdata, handles)
inf=get(hObject,'Value');
texto=get(hObject,'String');
switch inf
    case 1
        set(handles.txt1,'string',['UK','1973']);
    case 2
        set(handles.txt1,'string',['UK','1973']);
    case 3
        set(handles.txt1,'string',['ESP','1973']);
    case 4
        set(handles.txt1,'string',['USA','1973']);
    case 5
        set(handles.txt1,'string',['UK','1973']);
    case 6
        set(handles.txt1,'string',['EC','1234']);
    case 7
        set(handles.txt1,'string',['USA','1234']);
    case 8
        set(handles.txt1,'string',['ARG','1234']);
    case 9
        set(handles.txt1,'string',['GER','...']);
    case 10
        set(handles.txt1,'string',['USA','...']);
    case 11
        set(handles.txt1,'string',['FNL','...']);
    case 12
        set(handles.txt1,'string',['GER','1223']);
end
set(handles.text2,'String',texto(inf))
guidata(hObject,handles);
```

En la parte de inicialización del programa editamos las funciones del Listado 3.35.

Listado 3.35: Función de apertura *OpeningFcn*.

```
function Listbox_OpeningFcn(hObject, eventdata, handles,
   varargin)
set(handles.text2,'string','List Box');
set(handles.txt1,'string',['UK','1973']);
handles.output = hObject;
guidata(hObject, handles);
```

En el ejemplo de la Figura 3.26 asignaremos a una *lista 2* la posición de la *lista 1*. Se hará simplemente una conversión de potencia en watts a una potencia en decibeles.

El campo *tag* del primer list box es *lista1* y del segundo *lista2*. Las condiciones se presentan en el Listado 3.36.

Listado 3.36: Función de apertura *OpeningFcn*.

```
function Listbox_2_OpeningFcn(hObject, eventdata, handles,
   varargin)
```

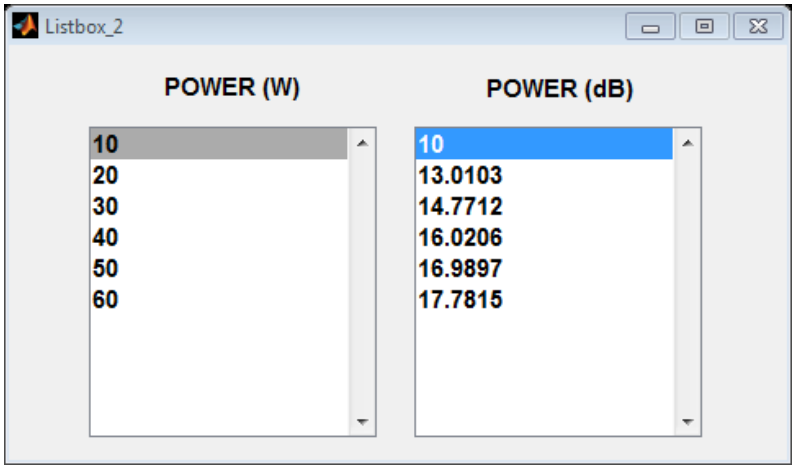

Figura 3.26: *Listbox* concurrente.

```
set(handles.lista1,'String',[10 20 30 40 50 60]);
set(handles.lista2,'String',[db(10,'Power') db(20,'Power')...
db(30,'Power') db(40,'Power') db(50,'Power')]);
handles.output = hObject;
guidata(hObject, handles);
```

La programación de la lista 1 se muestra en el Listado 3.37.

Listado 3.37: Función de la lista 1.

```
function lista1_Callback(hObject, eventdata, handles)
a=get(hObject,'Value');
set(handles.lista2,'Value',a);
```

3.8 Uso del *slider*.

Con este ejemplo se pretende mostrar cómo configurar el *slider*, así como el *check box*. En sí, este programa es un detector de nivel del dato ingresado por el usuario. (Ver Figura 3.27.)

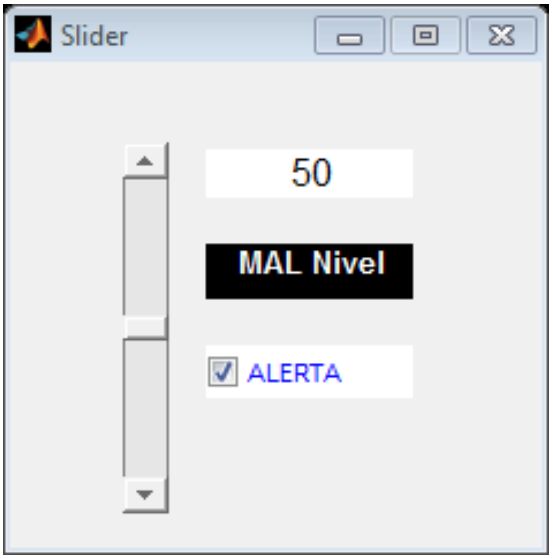

Figura 3.27: Componente *Slider* y *check box*.

Iniciamos un nuevo GUI con el nombre *slider*. Añadimos los componentes que muestra la Figura 3.27. Editamos el *checkbox* como muestra la Figura 3.28.

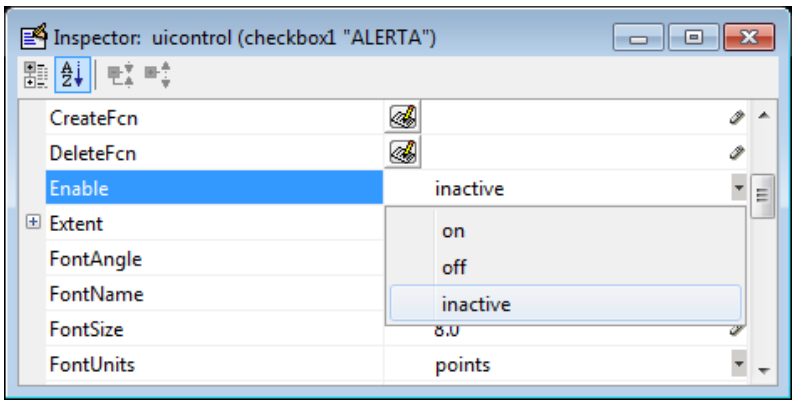

Figura 3.28: Componente *Slider*: desactivación.

Enable en estado *inactive* permite escribir sobre el *checkbox* y no poder modificar este estado por el usuario. La función del *slider* se muestra en el Listado 3.38.

Listado 3.38: Función del *slider*.

```
function slider1_Callback(hObject, eventdata, handles)
handles.slider1=get(hObject,'Value');
handles.slider1=100.*handles.slider1;
set(handles.text1,'String',handles.slider1);
if (handles.slider1 < 50)
    set(handles.text3,'String','BUEN Nivel');
    set(handles.checkbox1,'Value',0) ;
else
    set(handles.text3,'String','MAL Nivel');
    set(handles.checkbox1,'Value',1) ;
end
guidata(hObject, handles);
```

El conjunto de la secuencia *if* determina dos resultados:

1. Si el valor del *slider* es menor que 50 escribe en el componente *text3* el texto "Buen Nivel" y mantiene en cero el estado del *checkbox*.

2. Si el valor del *slider* es mayor o igual que 50 escribe en el componente *text3* el texto "Mal Nivel" y mantiene en uno el estado del *checkbox*.

Realicemos un programa más interesante con el uso del *slider*: cambiar el contraste de una imagen. En una nueva GUI colocamos un *slider*, un *push-button* y un *axes* como muestra la Figura 3.29.

El campo *tag* del botón es *open*, igual que el campo *string*. La programación es la misma que se usó para abrir una imagen. (Ver Listado 3.39.)

Listado 3.39: Función del botón.

```
function abrir_Callback(hObject, eventdata, handles)
[nombre,ruta]=uigetfile('*.jpg','Abrir imagen');
if isequal(nombre,0)
```

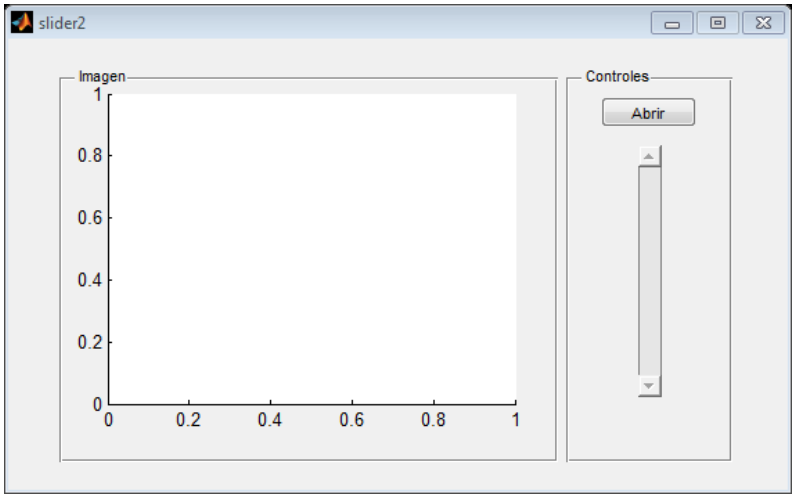

Figura 3.29: Cambio de contraste de una imagen con *slider*.

```
    return
else
    imagen=imread(strcat(ruta,nombre));
    image(imagen);
    set(handles.slider1,'Enable','on')
end
handles.imagen=imagen;
guidata(hObject,handles)
```

El campo *tag* del *slider* es el mismo que aparece por defecto. La programación del *slider* se muestra en el Listado 3.40.

Listado 3.40: Función del *slider*.

```
function slider1_Callback(hObject, eventdata, handles)
a=0.02*get(hObject,'Value');
imag_mod=rgb2gray(handles.imagen);
imag_mod=a*imag_mod;
image(imag_mod); colormap('gray')
```

Para evitar errores al mover el *slider* cuando no se ha cargado ninguna imagen, en la parte de las condiciones iniciales del programa se coloca el campo *enable* del *slider* a *off*. (Ver Listado 3.41.)

Listado 3.41: Función del apertura.

```
function slider2_OpeningFcn(hObject,eventdata,handles,
   varargin)
set(handles.slider1,'Enable','off')
handles.output = hObject;
guidata(hObject, handles);
```

El programa en funcionamiento se muestra en la Figuras 3.30.

Los valores máximos y mínimos del *slider* se modifican en el campo *max* y *min* del *Property Inspector*. El valor inicial (desde donde arranca el *slider*) se modifica en el campo *value*.

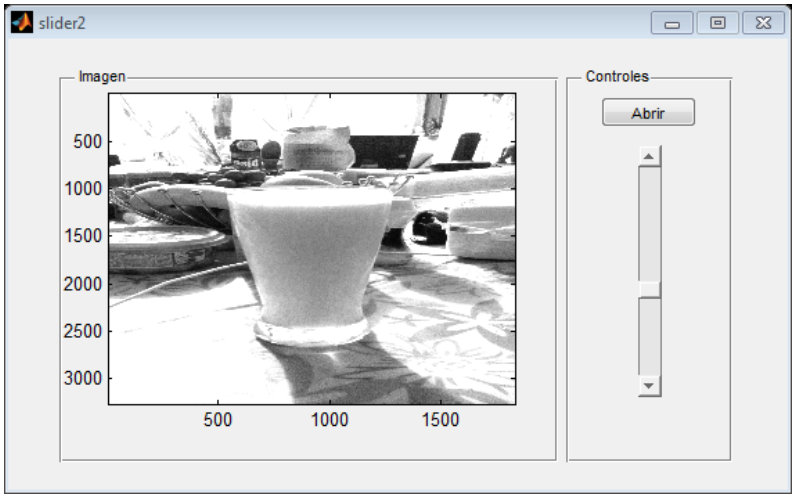

Figura 3.30: Resultado del cambio de contraste con *slider*.

Debido a que este elemento tiene dos modos de cambio: las flechas tanto inferior como superior y la barra de deslizamiento, se establece el valor de cambio en el campo *Slider Step* del *Property Inspector*. En el ejemplo, si queremos cambiar con las flechas el valor del *slider* de uno en uno y el valor del mismo de dos en dos con la barra de deslizamiento lo hacemos configurando la matriz de *Slider Step* así: [0.05 0.1].

¿Por qué estos valores? La parte *x* corresponde al porcentaje de cambio de las flechas y la parte *y* corresponde al porcentaje de cambio de la barra deslizante. Para encontrar estos valores se calcula:

Listado 3.42: Valor de cambio del *slider*.

```
Rango * x = valor de cambio de las flechas
x= valor de cambio de las flechas / Rango
Rango * y = valor de cambio de la barra
y= valor de cambio de la barra / Rango
```

Con los datos de este ejemplo, la configuración queda conforme mostrado en el Listado 3.43.

Listado 3.43: Valor de cambio del *slider*.

```
Rango = Valor mayor - Valor menor = 10-(-10) = 20
Valor de cambio de las flechas = 1
Valor de cambio de la barra = 2
X=1/20=0.05
Y=2/20=1
```

Para grabar la imagen modificada en disco, se crea un nuevo botón usando la opción de duplicar el botón que abre la imagen. Para hacer esto, basta dar clic derecho sobre el botón *Abrir* y seleccionar *duplicate* o bien el comando ctrl+d. En este nuevo botón, editamos el campo *tag* y el campo *string* como *guardar*. Añadimos un par de líneas de código en la función del *slider*, de forma a exportar la imagen modificada hacia la función del botón de guardado. Esta modificación se presenta en el Listado 3.44, específicamente en las dos última líneas, donde guardamos la imagen en un *handles* para que esta matriz sea leída en la función del botón guardar.

Listado 3.44: Exportar imagen modificada por el *slider*.

```
function slider1_Callback(hObject, eventdata, handles)
a=0.02*get(hObject,'Value');
imag_mod=rgb2gray(handles.imagen);
imag_mod=a*imag_mod;
image(imag_mod);
colormap('gray')
handles.imag_mod=imag_mod;
guidata(hObject,handles)
```

Al igual que tenemos una función para leer la ruta y nombre de un archivo que deseamos abrir, podemos usar la función *uiputfile* para seleccionar la carpeta y el nombre del archivo o imagen que vamos a grabar en disco. De esa forma, usando esta función, colocamos el código del Listado 3.45 dentro de la función del nuevo botón creado para guardar la imagen. Observaremos que se ha creado un nuevo archivo de imagen en la carpeta seleccionada.

Listado 3.45: Guardar imagen modificada por el *slider*.

```
[nombre,ruta] = uiputfile('*.jpg','Guardar imagen jpg');
if nombre~=0
    ruta_nombre=strcat(ruta,nombre);
    imag_mod=handles.imag_mod;
    imwrite(imag_mod,ruta_nombre);
end
```

3.9 Colocar una imagen en un botón.

La interfaz gráfica de usuario de Matlab permite personalizar la presentación de los botones como lo muestra la Figura 3.31. Para lograr esto, editamos el código del Listado 3.46 en la parte del m-file destinada a la inicialización del programa. Observar que será necesario modificar la ruta donde se encuentra la imagen, así como el nombre de la misma.

Listado 3.46: Función del apertura.

```
function etiqbutton_OpeningFcn(hObject, eventdata, handles,
   varargin)
ruta_img='C:\matlab_img\Etiqbutton_img';
[a,map]=imread([ruta_img,'\vol.jpg']);
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton1,'CData',g);

[a,map]=imread([ruta_img,'\stop.jpg']);
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.pushbutton2,'CData',g);
```

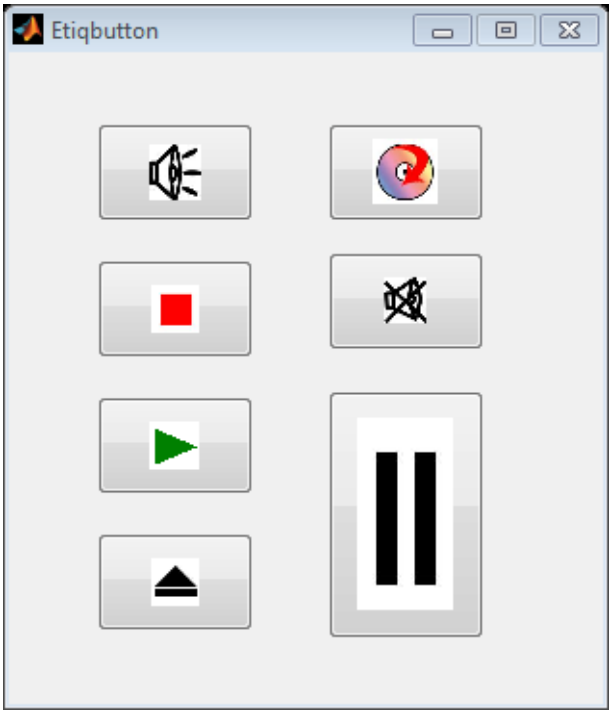

Figura 3.31: Etiquetación de botones.

```
18
19 [a,map]=imread([ruta_img,'\play.jpg']);
20 [r,c,d]=size(a);
21 x=ceil(r/30);
22 y=ceil(c/30);
23 g=a(1:x:end,1:y:end,:);
24 g(g==255)=5.5*255;
25 set(handles.pushbutton3,'CData',g);
26
27 [a,map]=imread([ruta_img,'\open_files.jpg']);
28 [r,c,d]=size(a);
29 x=ceil(r/30);
30 y=ceil(c/30);
31 g=a(1:x:end,1:y:end,:);
32 g(g==255)=5.5*255;
33 set(handles.pushbutton4,'CData',g);
34
35 [a,map]=imread([ruta_img,'\cd_eject.jpg']);
36 [r,c,d]=size(a);
37 x=ceil(r/35);
38 y=ceil(c/35);
39 g=a(1:x:end,1:y:end,:);
40 g(g==255)=5.5*255;
41 set(handles.pushbutton5,'CData',g);
42
43 [a,map]=imread([ruta_img,'\pause.jpg']);
44 [r,c,d]=size(a);
45 x=ceil(r/100);
46 y=ceil(c/80);
47 g=a(1:x:end,1:y:end,:);
48 g(g==255)=5.5*255;
49 set(handles.pushbutton6,'CData',g);
```

```
50
51 [a,map]=imread([ruta_img,'\mute2.jpg']);
52 [r,c,d]=size(a);
53 x=ceil(r/30);
54 y=ceil(c/30);
55 g=a(1:x:end,1:y:end,:);
56 g(g==255)=5.5*255;
57 set(handles.pushbutton7,'CData',g);
58 % Choose default command line output for etiqbutton
59 handles.output = hObject;
60 % Update handles structure
61 guidata(hObject, handles);
```

Para personalizar aún más, se puede hacer aparecer el nombre del botón cuando se acerca el cursor, simplemente llenando el campo *TooltipString* en el *Property Inspector*.

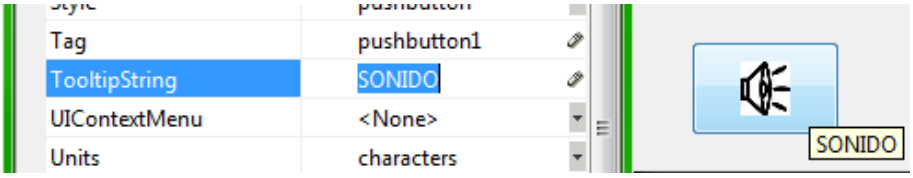

Figura 3.32: *TooltipString* del botón de sonido.

3.10 Imágenes en diferentes axes.

Con este sencillo programa se mostrará como asignar a cada axes de un GUI un gráfico o imagen específica, conforme mostrado en la Figura 3.33. La función a utilizar será *axes*, cuyo argumento es el manejador o *handles* de componente de graficación del archivo .fig.

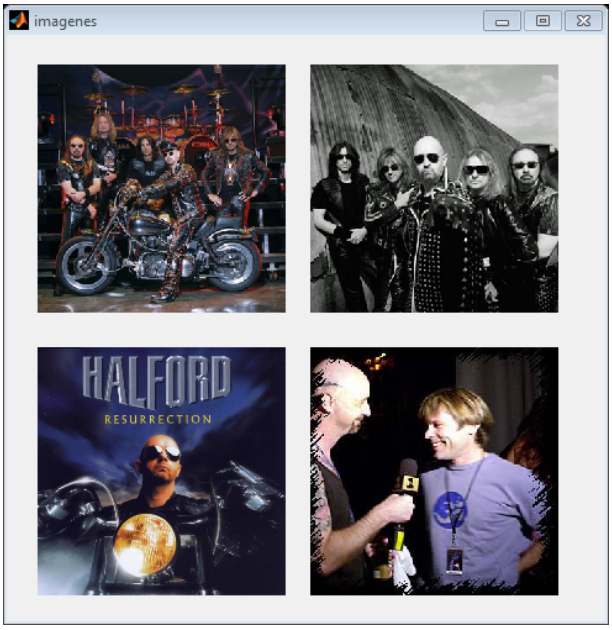

Figura 3.33: GUI con varias figuras.

En la parte de inicialización del programa editamos él código del Listado 3.47. Es necesario tener las imágenes dentro de la misma carpeta donde está el código o bien

modificar la ruta donde éstas se encuentran. Los nombres de cada imagen son *una*, *dos*, *tres* y *cuatro*.

Listado 3.47: Función de apertura.

```
function imagenes_OpeningFcn(hObject, eventdata, handles,
   varargin)
ruta_img='C:\matlab_img\imagenes_img';
axes(handles.axes1)
background = imread([ruta_img,'\una.jpg']);
image(background);
axis off;

axes(handles.axes2)
background = imread([ruta_img,'\dos.jpg']);
image(background);
axis off;

axes(handles.axes3)
background = imread([ruta_img,'\tres.jpg']);
image(background);
axis off;

axes(handles.axes4)
background = imread([ruta_img,'\cuatro.jpg']);
image(background);
axis off;

handles.output = hObject;
guidata(hObject, handles);
```

3.11 Segmentación de imagen *puzzle*.

Cada imagen leída en Matlab representa un matriz de tres dimensiones. A partir de esta matriz, se puede dividir la imagen en otra matrices de cualquier dimensión. Una aplicación que mostrará esa segmentación es el programa *puzzle*, que dividirá la imagen en cuatro matrices y las unirá de forma a generar una nueva imagen. La GUI consta de tres botones: uno para abrir la imagen, otro botón para segmentar la imagen y formar la nueva matriz modificada y un botón para guardar la nueva imagen. En entorno de esta GUI se muestra en la Figura 3.34.

En el Listado 3.48 se muestra el código que abre la ventana de selección de imagen.

Listado 3.48: Función para seleccionar la imagen.

```
function abrir_imagen_Callback(hObject, eventdata, handles)
[nombre, ruta]=uigetfile('*.jpg','Abrir imagen');
if ruta ~=0
    imagen=imread(strcat(ruta,nombre));
    dim_f=size(imagen,1);
    imagen=imresize(imagen,[dim_f dim_f]);
    imshow(imagen)
    handles.imagen=imagen;
end
guidata(hObject, handles);
```

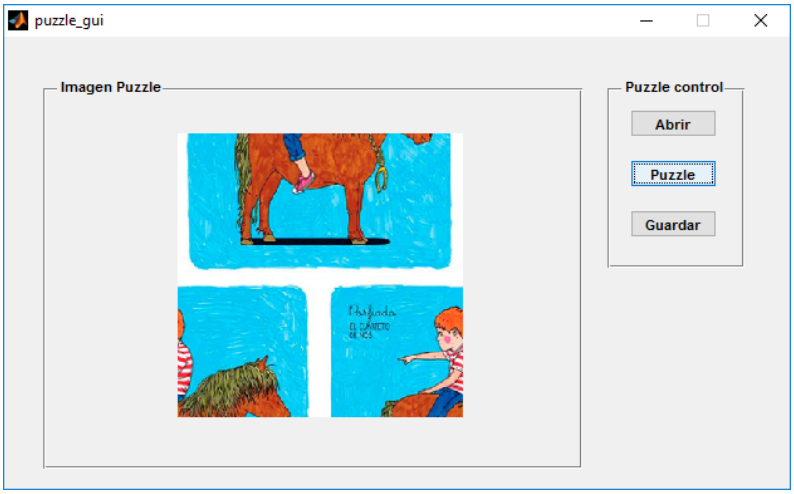

Figura 3.34: Programa *puzzle*: segmentación de imagen.

En el Listado 3.49 se muestra el código que abre la ventana de selección de imagen.

Listado 3.49: Función de segmentación de la imagen y concatenación.

```
function puzzle_imagen_Callback(hObject, eventdata, handles)
% Llamar imagen leida
imagen=handles.imagen;
dim_f=size(imagen,2);
if mod(dim_f,2)==1
    dim_f=dim_f-1;
end
imagen=imresize(imagen,[dim_f dim_f]);
num_imagenes=2;
% Cambiar dimension
pix_f=floor(dim_f/num_imagenes);

% Segmentar en 4 imagenes
% 1
img_r=imagen(1:pix_f,1:pix_f,1);
img_g=imagen(1:pix_f,1:pix_f,2);
img_b=imagen(1:pix_f,1:pix_f,3);
img_trozo1(:,:,1)=img_r;
img_trozo1(:,:,2)=img_g;
img_trozo1(:,:,3)=img_b;

% 2
img_r=imagen(pix_f+1:end,1:pix_f,1);
img_g=imagen(pix_f+1:end,1:pix_f,2);
img_b=imagen(pix_f+1:end,1:pix_f,3);
img_trozo2(:,:,1)=img_r;
img_trozo2(:,:,2)=img_g;
img_trozo2(:,:,3)=img_b;

% 3
img_r=imagen(1:pix_f,pix_f+1:end,1);
img_g=imagen(1:pix_f,pix_f+1:end,2);
img_b=imagen(1:pix_f,pix_f+1:end,3);
img_trozo3(:,:,1)=img_r;
img_trozo3(:,:,2)=img_g;
```

```
img_trozo3(:,:,3)=img_b;

% 4
img_r=imagen(pix_f+1:end,pix_f+1:end,1);
img_g=imagen(pix_f+1:end,pix_f+1:end,2);
img_b=imagen(pix_f+1:end,pix_f+1:end,3);
img_trozo4(:,:,1)=img_r;
img_trozo4(:,:,2)=img_g;
img_trozo4(:,:,3)=img_b;

super_imagen(:,:,:,1)=img_trozo1;
super_imagen(:,:,:,2)=img_trozo2;
super_imagen(:,:,:,3)=img_trozo3;
super_imagen(:,:,:,4)=img_trozo4;
% Ordenar las cuatro imagenes aleatoriamente
orden=randperm(4);

final_image=uint8(zeros(size(imagen)));

final_image(1:pix_f,1:pix_f,:)=...
super_imagen(:,:,:,orden(1));
final_image(pix_f+1:end,1:pix_f,:)=...
super_imagen(:,:,:,orden(2));
final_image(1:pix_f,pix_f+1:end,:)=...
super_imagen(:,:,:,orden(3));
final_image(pix_f+1:end,pix_f+1:end,:)=...
super_imagen(:,:,:,orden(4));
% Mostrar imagen
imshow(final_image)
% Exportarla a la funcion de guardado
handles.final_image=final_image;
% Actualizar el handles
guidata(hObject, handles)
```

Finalmente, en el Listado 3.50 se muestra el código que abre la ventana que permite guardar la imagen modificada en disco.

Listado 3.50: Función para guardar la imagen.

```
function guardar_Callback(hObject, eventdata, handles)
[nombre, ruta]=uiputfile('*.jpg','Guardar imagen');
if nombre ~=0
    final_image=handles.final_image;
    imwrite(final_image,strcat(ruta,nombre))
end
```

3.12 Panel de botones y botón de estado.

Con este programa se pretende mostrar el uso de *toggle button*, así como el empleo de *panel button*. Una de las novedades que se presenta a partir de la versión 7.0.1 de MATLAB es el panel de botones (*button panel*). En este ejemplo, los botones del panel deben tener la condición de que solo uno de ellos deber estar en alto y los demás en estado bajo, para así poder configurar el estilo de línea. Además, el botón encendido

debe permanecer es estado alto aunque se lo vuelva a seleccionar.

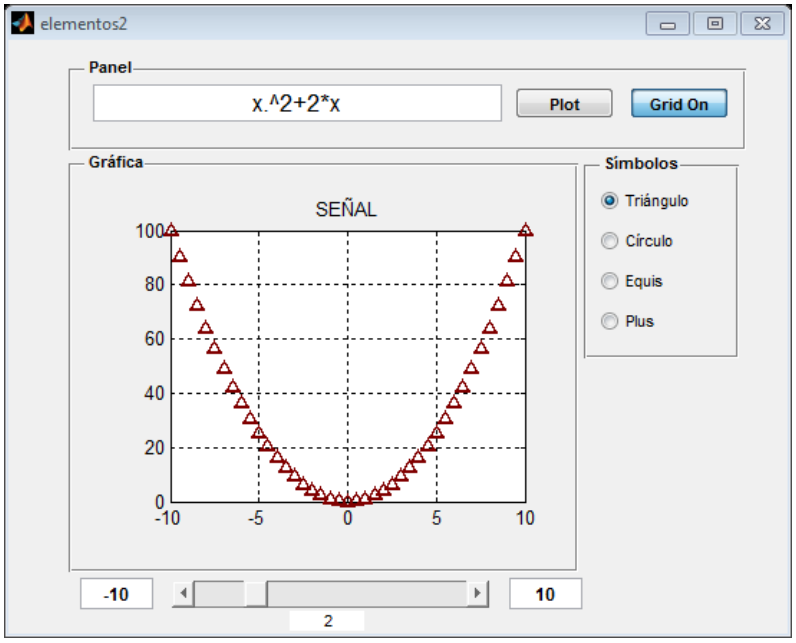

Figura 3.35: Entorno de la GUI.

La configuración de los parámetros de la función a graficar se basa en la Tabla 3.2.

Tabla 3.2: Estilos para gráficas: línea, color y marca

Color		Símbolo		Línea	
b	azul	**.**	punto	-	continua
g	verde	**o**	círculo	:	punteada
r	rojo	**x**	marca x	.-	punto raya
c	cyan	**+**	más	–	entrecortada
m	magenta	*	estrella		
y	amarillo	**s**	cuadrado		
k	negro	**d**	diamante		
		v	triángulo abajo		
		^	triángulo arriba		
		<	triángulo izquierda		
		>	triángulo derecha		
		p	pentagrama		
		h	hexagrama		

El sector del archivo .m destinado a la programación de las condiciones iniciales tiene la programación mostrada en el Listado 3.51.

Listado 3.51: Función del apertura.

```
1 function elementos2_OpeningFcn(hObject, eventdata, handles,
     varargin)
2 axes(handles.grafica)
3 x=-10:0.5:10;
```

```
handles.x=x;
handles.h=plot(x,x.^2);
set(handles.h,'LineStyle','^','Color',[0.5 0 0]);
grid off;
set(handles.grids,'String','Grid Off');
title ('SIGNAL');
handles.slider1=0.1;
set(handles.text1,'String',handles.slider1);
handles.output = hObject;
guidata(hObject, handles);
```

Una vez que se ha colocado cada uno de los botones dentro del panel de botones y se los ha etiquetado, se ejecuta lo que muestra la Figura 3.36.

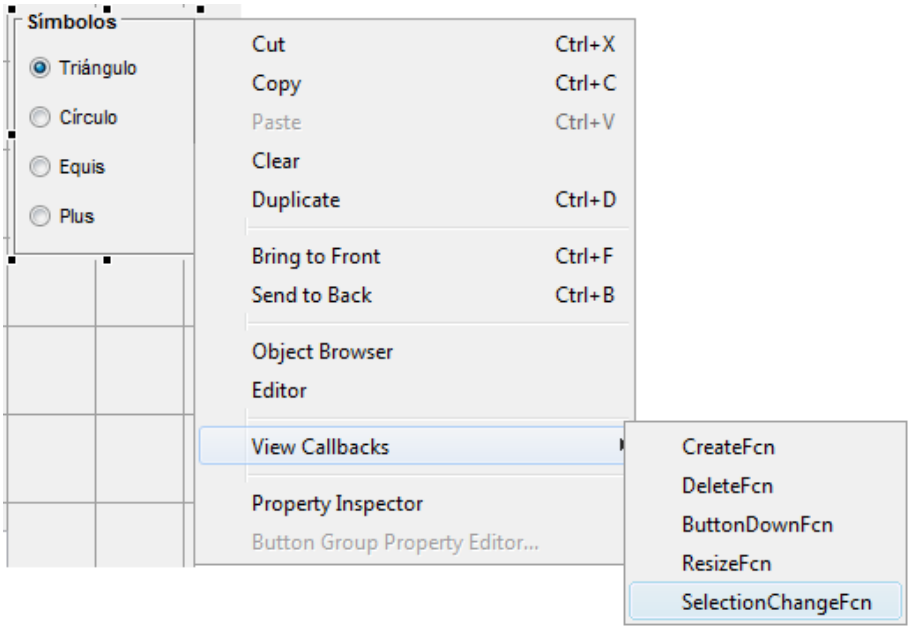

Figura 3.36: Ubicación en la función *Selection Change*.

En el archivo .m asociado se creará el código del Listado 3.52.

Listado 3.52: Función *Selection Change*.

```
function uipanel1_SelectionChangeFcn(hObject, eventdata,
    handles)
% hObject handle to uipanel1 (see GCBO)
% eventdata reserved - to be defined in a future version of
    MATLAB
% handles structure with handles and user data (see GUIDATA)
```

Para este ejemplo, el campo *tag* de de cada botón del panel se ha etiquetado uno, dos, tres y cuatro. De esta forma, es más fácil trabajar cada configuración en el archivo .m asociado. La programación del panel de botones se muestra en el Listado 3.53.

Listado 3.53: Función *Selection Change*.

```
function uipanel1_SelectionChangeFcn(hObject, eventdata,
    handles)
if (hObject==handles.uno)
```

```
    set(handles.h,'LineStyle','.','Color','g');
elseif(hObject==handles.dos)
    set(handles.h,'LineStyle','o','Color','r');
elseif(hObject==handles.tres)
    set(handles.h,'LineStyle','x','Color','c');
else
    set(handles.h,'LineStyle','h','Color',[0.52 0 0]);
end
```

La función del *slider* se muestra en el Listado 3.54.

Listado 3.54: Función del *slider*.

```
function slider1_Callback(hObject, eventdata, handles)
valor=get(hObject,'Value');
valor= valor*10;
if valor==0
    valor=valor+0.01;
end
set(handles.h,'LineWidth',valor);
set(handles.text1,'String',valor);
guidata(hObject,handles);
```

El valor por defecto del *slider* va de 0 a 1. Por esta razón se lo multiplica por 10. Como el valor capturado del *slider* se usa para el ancho de línea de la gráfica, este valor no deber ser cero y es por eso que usamos una sentencia *if*, para que cuando el valor ingresado sea cero, inmediatamente se le sume 0.01.

La función *set(handles.h,'LineWidth',handles.slider1)* modifica el valor del ancho de línea de la señal. La función *set(handles.edit1,'String',handles.slider1)* escribe el valor numérico en *edit1*. La configuración del *toggle button* (etiquetado como *grids*) que añade el *grid*, tiene el código del Listado 3.55.

Listado 3.55: Función del activación y desactivación del *grid*.

```
function grids_Callback(hObject, eventdata, handles)
estado=get(hObject,'Value');
if estado==1
    grid on;
    set(handles.grids,'String','Grid On');
else
    grid off;
    set(handles.grids,'String','Grid Off');
end
```

La sentencia *set(handles.grids,'String','GRID ON')* modifica el texto del *toggle button*, de esta forma se leerá *GRID ON* o *GRID OFF* dependiendo de la selección del botón. Al ingresar la función a graficar, el botón *plot* ejecutará el código del Listado 3.56.

Listado 3.56: Función del graficación.

```
function boton_plot_Callback(hObject, eventdata, handles)
axes(handles.grafica)

men=str2double(get(handles.menor,'String'));
```

```
may=str2double(get(handles.mayor,'String'));

x=men:0.5:may;

ecuacion=get(handles.ecuacion_txt,'String');
ecuacion=char(ecuacion);

jud=get(handles.h,'Marker');
asp=get(handles.h,'LineWidth');
rie=get(handles.h,'Color');
handles.h=plot(x,eval(ecuacion));
set(handles.h,'LineStyle',jud);
set(handles.h,'LineWidth',asp);
set(handles.h,'Color',rie);

handles.die=get(handles.grids,'Value');
if handles.die==1
    grid on;
    set(handles.grids,'String','GRID ON');
else
    grid off;
    set(handles.grids,'String','GRID OFF');
end
guidata(hObject,handles)
```

3.13 Lectura y escritura de un archivo de Excel.

Usando las funciones *uigetfile* y *uiputfile* en combinación con *xlsread* y *xlswrite* podemos crear una GUI para leer datos de un archivo de excel, procesarlos en Matlab y luego guardar el resultado de ese procesamiento en otro archivo de excel. Este programa, una vez leídos los datos de una determinada columna del archivo excel, buscará los valores que estén por debajo de un umbral determinado por un *slider*. La GUI de este programa se presenta en la Figura 3.37.

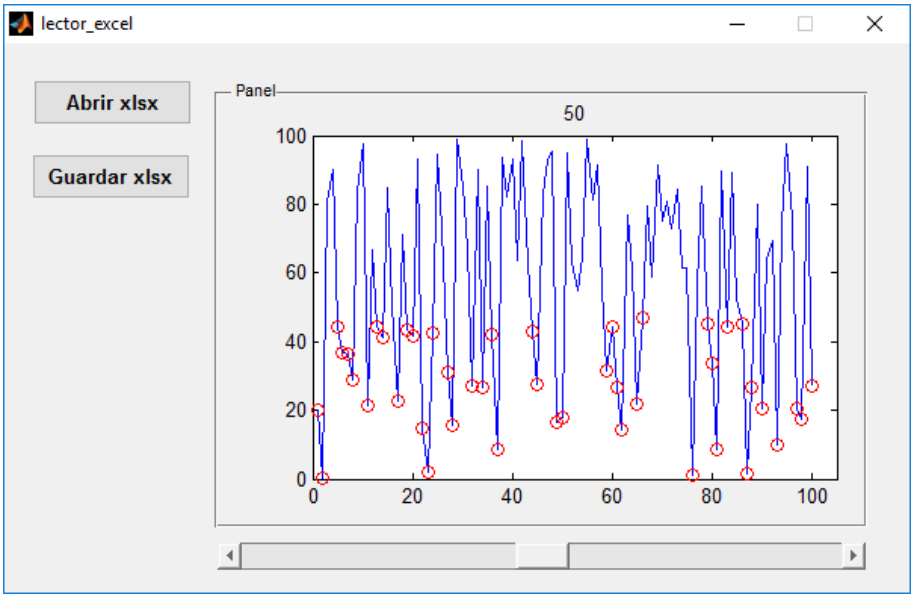

Figura 3.37: GUI para leer datos de excel.

El archivo de excel que con los datos a leer tendrá tres columnas de datos: una para enumeración y las dos restantes con datos provenientes de alguna medición. La columna que nos interesa es la primera luego de los datos de etiquetación. El formato del archivo se presenta en la Figura 3.38.

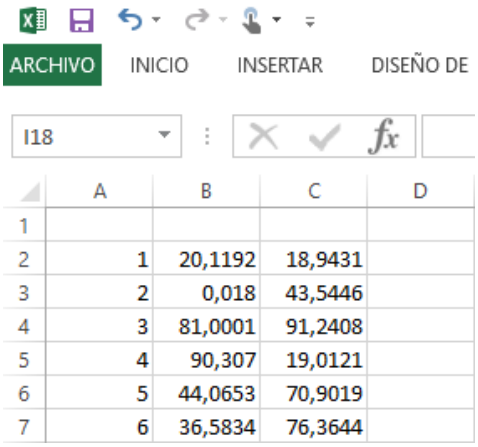

Figura 3.38: Formato de la hoja de excel a ser leída.

El procesamiento de los datos que realizará esta GUI es encontrar aquellos valores que estén por debajo de un cierto umbral y guardar esos datos en un archivo de excel. En este programa, se utilizará la opción de habilitación tanto del botón de guardar como del *slider* que marca el valor del umbral debido a que si no están disponibles los datos no se ejecute el código de búsqueda y almacenamiento. Esta opción de habilitación puede ser configurada tanto en el archivo .m (en la función de apertura o condiciones iniciales) como en el archivo .fig. Ver Listado 3.57.

Listado 3.57: Función de apertura.

```
function lector_excel_OpeningFcn(hObject, eventdata, handles,
    varargin)
set(handles.slider1,'Enable','off');
handles.output = hObject;
guidata(hObject, handles);
```

El código que se ejecuta al presionar el botón de lectura del archivo se muestra en el Listado 3.58. En la función *uigetfile* colocamos la extensión del archivo excel, así como una condición de control con la función *if* en caso de que el usuario seleccione la opción de cancelar. También, activamos el slider para que puede ser manipulado y establezca el umbral de comparación. La función xlsread leerá los datos de la hoja 1 desde la posición B2 hasta las posición B101. (Ese rango corresponde a los datos que están escritos en el archivo de excel.) Una vez graficados los datos, guardamos este vector leído en un *handles* para que pueda ser procesado por la función del *slider*.

Listado 3.58: Código del botón para leer datos.

```
function pushbutton1_Callback(hObject, eventdata, handles)
[nombre,ruta]=uigetfile('*.xlsx','Abrir excel');
if nombre~=0
    set(handles.slider1,'Enable','on');
    set(handles.pushbutton2,'Enable','on');
    filename = strcat(ruta,nombre);
    hoja = 1;
```

```
    xlRange1a = 'B2:B101';
    datos_a = xlsread(filename,hoja,xlRange1a);
    plot(datos_a)
    xlim([0,105])
    handles.datos_a=datos_a;
    guidata(hObject,handles)
end
```

El *slider* trabajará con los datos obtenidos por la función del Listado 3.58, que fueron almacenados por el *handles*. El *slider* marcará un umbral de toma de decisión, en el cual los datos (que van en un rango de 1 hasta 100) serán comparados. Si uno de los datos leídos está por debajo del umbral establecido por el *slider*, se marcará esta coordenada en la gráfica y se guardará en una variable que también será almacenada en un *handles*. El código de este procesamiento se presenta en el Listado 3.59.

Listado 3.59: Función del *slider*.

```
function slider1_Callback(hObject, eventdata, handles)
datos_a=handles.datos_a;
plot(datos_a)
xlim([0,105])
val_slider=100*get(handles.slider1,'Value');
[pos_umbral]=find(datos_a<val_slider);
hold on
plot(pos_umbral,datos_a(pos_umbral),'or')
title(num2str(val_slider))
hold off
handles.pos_umbral=pos_umbral;
guidata(hObject,handles)
```

El botón para guardar datos usa la función *uiputfile* para que el usuario escoja el directorio y el nombre del archivo excel. Igual que en el caso anterior, una función *if* sirve para el caso de que el usuario presione el botón de cancelar. Determinamos la dimensión de los datos con la función *length* y formamos el *string* del rango para la hoja de excel. Con la función *xlswrite* almacenamos los datos en el archivo de excel. El código de este procesamiento se muestra en el Listado 3.60.

Listado 3.60: Función guarda datos.

```
function pushbutton2_Callback(hObject, eventdata, handles)
[nombre,ruta]=uiputfile('*.xlsx','Archivo guardar');
if nombre ~=0
    pos_umbral=handles.pos_umbral;
    ini = 'B2';
    fin = ['B',num2str(length(pos_umbral))];
    archivo=strcat(ruta,nombre);
    xlswrite(archivo,handles.datos_a(pos_umbral),1,[ini,':',
       fin]);
end
```

3.14 Control de Simulink desde GUI.

Con este programa, mostrado en la Figura 3.39, se pretende mostrar la interacción entre una Interfaz Gráfica y Simulink, que es un entorno de programación gráfica, donde los

archivos generados se denominan modelos.

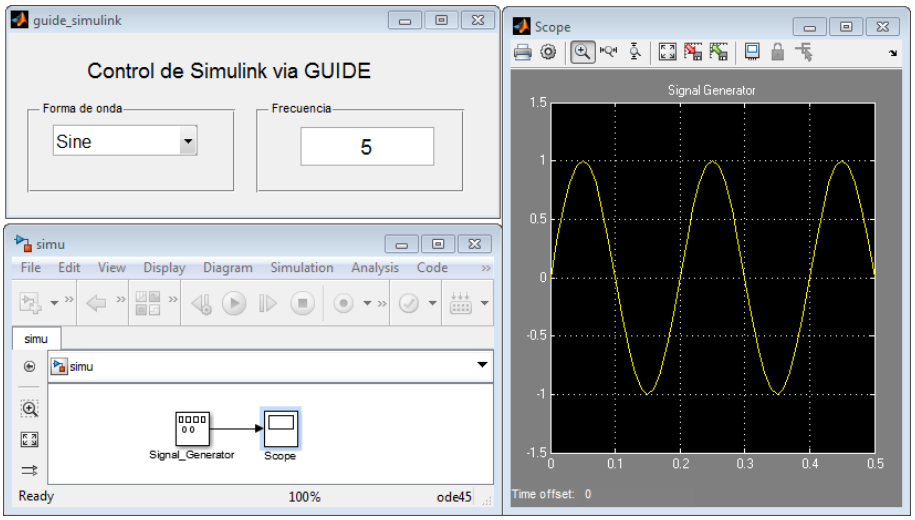

Figura 3.39: Simulink controlado por GUI.

Este sencillo programa controla dos parámetros del bloque *Signal_Generator*: la forma de onda y la frecuencia.

En la parte de inicialización del programa se escribe el código de Listado 3.61.

Listado 3.61: Función del apertura.

```
1 function guide_simulink_OpeningFcn(hObject, eventdata,
     handles, varargin)
2 find_system('Name','simu');
3 open_system('simu');
4 set_param('simu/Signal_Generator','WaveForm','sine');
5 set_param('simu/Signal_Generator','frequency','5');
6 set_param(gcs,'SimulationCommand','Start');
7 handles.output = hObject;
8 guidata(hObject, handles);
```

Estos comandos son los primeros que se ejecutan al abrir la interfaz gráfica. *Find_system* y *open_system* son funciones para comprobar si existe el programa en simulink y abrirlo. La sentencia para escribir en los bloques de simulink es *set_param*, que se usa para establecer en el parámetro *wave form* del bloque *Signal_Generator* la onda *sine*. La sintaxis de *set_param es*:

set_param('nombre_del_programa/nombre_del_bloque','parámetro','valor')

Para obtener el nombre de los parámetros de los bloques de Simulink que se modificarás desde la GUI de Matlab, se lo puede hacer con la siguiente sentencia, modificada aquí para el caso particular un bloque PID:

p = get_param('modelo/Signal Generator', 'DialogParameters')

Como resultado, se obtiene en la variable *p* una estructura mostrada en el Listado 3.62, la cual contiene la característica de cada parámetro del bloque.

Listado 3.62: Estructura con parámetros de Simulink.

```
p =
             WaveForm: [1x1 struct]
          TimeSource: [1x1 struct]
           Amplitude: [1x1 struct]
           Frequency: [1x1 struct]
               Units: [1x1 struct]
    VectorParams1D: [1x1 struct]
```

Para este bloque si se desea cambiar el valor del parámetro Proportional, debe ser de la siguiente manera:

set_param('modelo/Signal Generator', 'Frequency','10')

El comando *set_param(gcs,'SimulationCommand','Start')* es para iniciar la ejecución del programa en simulink.

La programación del *pop up* menú (etiquetada el en campo *tag* como *wave*) se presenta en el Listado 3.63. La caja de texto donde se ingresa la frecuencia está etiquetada como *frec*.

Listado 3.63: Función que modifica la forma de onda del bloque de Simulink.

```
function wave_Callback(hObject, eventdata, handles)
onda=get(hObject,'Value');
frec=get(handles.frec,'String');
set_param('simu/Signal_Generator','frequency',frec);
if onda==1
    set_param('simu/Signal_Generator','WaveForm','sine');
elseif onda==2
    set_param('simu/Signal_Generator','WaveForm','square');
elseif onda==3
    set_param('simu/Signal_Generator','WaveForm','sawtooth');
else
    set_param('simu/Signal_Generator','WaveForm','random');
end
set_param(gcs,'SimulationCommand','Start');
```

Otro programa que controla un modelo de Simulink desde una GUI de Matlab es el mostrado en la Figura 3.40.

Como se ve en la figura precedente, los valores a ser sumados se ingresan desde una GUI y son evaluados en Simulink. La parte de condiciones iniciales se presenta en el Listado 3.64.

Listado 3.64: Función del apertura.

```
function suma_sim_OpeningFcn(hObject, eventdata, handles,
   varargin)
find_system('Name','sumar');
open_system('sumar');
set_param(gcs,'SimulationCommand','Start');
handles.output = hObject;
guidata(hObject, handles);
```

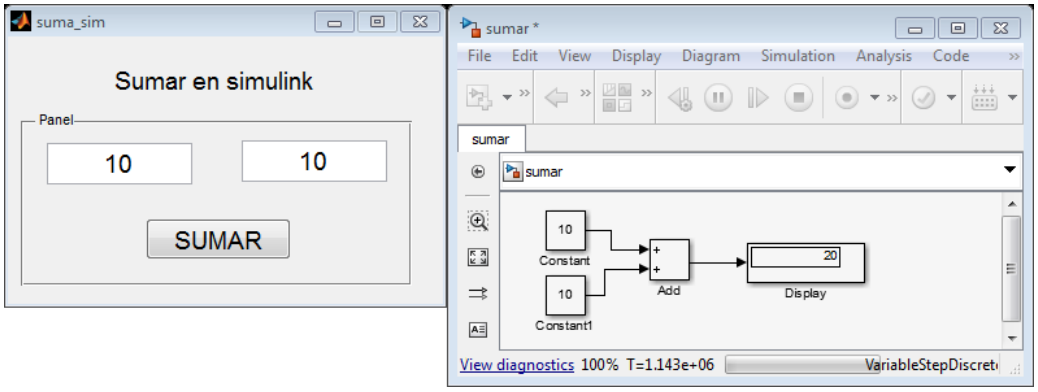

Figura 3.40: Suma de números en Simulink ingresados desde Matlab.

La programación del botón se muestra en el Listado 3.65.

Listado 3.65: Función del apertura.

```
function sumar_Callback(hObject, eventdata, handles)
val_a=get(handles.valor_a,'String');
val_b=get(handles.valor_b,'String');
set_param('sumar/Constant','Value',val_a);
set_param('sumar/Constant1','Value',val_b);
```

Para finalizar con los programas de interacción entre GUIDE y Simulink, se realizará un simple reproductor de audio, en el cual se podrá ejecutar play, pausa, continuar y stop; como también tener control del volumen de la reproducción. La Figura 3.41 muestra el entorno de este sencillo programa.

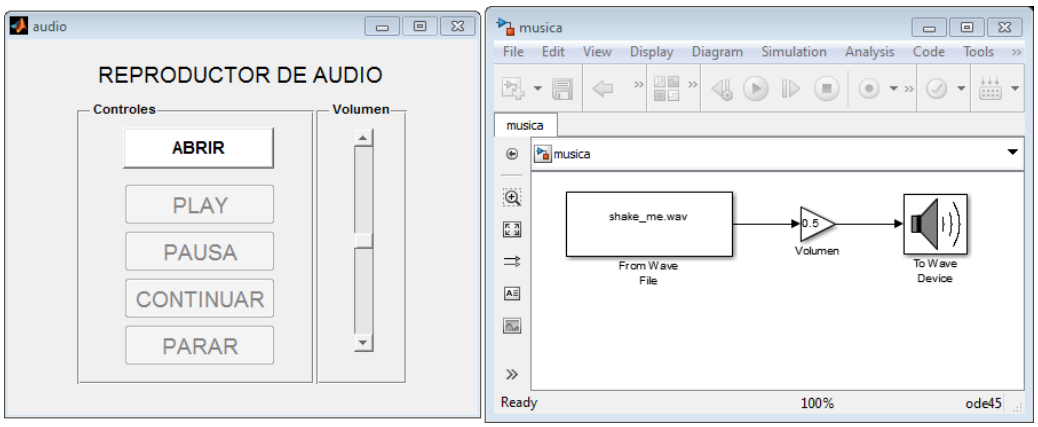

Figura 3.41: Reproductor de audio.

Los componentes del programa *musica* están en la librería *Signal Processing Blockset*. Es necesario mantener la etiqueta de cada bloque de Simulink.

Las condiciones iniciales de la GUI audio se presentan en el Listado 3.66.

Listado 3.66: Función del apertura.

```
function audio_OpeningFcn(hObject, eventdata, handles,
   varargin)
set(handles.volumen,'Value',0.5);
find_system('Name','musica');
```

```
4  open_system('musica');
5  set_param('musica/Volumen','Gain','0.5');
6  set(handles.play,'Enable','off');
7  set(handles.pause,'Enable','off');
8  set(handles.stopp,'Enable','off');
9  set(handles.contin,'Enable','off');
10 handles.output = hObject;
11 guidata(hObject, handles);
```

La línea 1 establece el valor de 0.5 al *slider*, que al abrir el programa estará justo en la mitad. Las líneas 2 y 3 son para encontrar el programa y abrirlo. La línea 4 establece en 0.5 el valor del parámetro *gain* del bloque *Volumen* en Simulink. La línea 5 toma el nombre del archivo de audio (*.wav) que se reproducirá (recuérdese que el archivo de audio debe estar en la misma carpeta del programa) y la línea 6 asigna este nombre al bloque *From Wave File*.

El fin de este programa, más allá de reproducir audio, es controlar la ejecución, pausa, continuación y parada de un programa en simulink desde una GUI. El botón que selecciona el archivo *wave* tiene la programación mostrada en el Listado 3.67.

Listado 3.67: Función del apertura.

```
1  function cancion_Callback(hObject, eventdata, handles)
2  [nomb,dire]=uigetfile('*.wav','Seleccione archivo');
3  if nomb==0
4      return
5  end
6  handles.cancion=fullfile(dire,nomb);
7  set(handles.play,'Enable','on');
8  set(handles.pause,'Enable','on');
9  set(handles.stopp,'Enable','on');
10 set(handles.contin,'Enable','on');
11 guidata(hObject,handles)
```

El campo *tag* de los botones play, pausa, continuar y parar es *play*, *pause*, *contin* y *stopp* respectivamente. La programación de cada botón se presenta en el Listado 3.68.

Listado 3.68: Función del apertura.

```
1  function play_Callback(hObject, eventdata, handles)
2  musica=handles.cancion;
3  set_param('musica/From Wave File','FileName',char(musica));
4  set_param(gcs,'SimulationCommand','Start');
5  function pause_Callback(hObject, eventdata, handles)
6  set_param(gcs,'SimulationCommand','Pause')
7  function stopp_Callback(hObject, eventdata, handles)
8  set_param(gcs,'SimulationCommand','Stop')
9  function contin_Callback(hObject, eventdata, handles)
10 set_param(gcs,'SimulationCommand','Continue')
```

Para el control del volumen se ha escogido un valor mínimo de 0 y un máximo de 1, que son los valores por defecto del *slider*. El *slider* se ha etiquetado (*tag*) como volumen. La programación del *slider* del volumen se muestra en el Listado 3.69.

Listado 3.69: Función del apertura.

```
function volumen_Callback(hObject, eventdata, handles)
volu=get(hObject,'Value');
set_param('musica/Volumen','Gain',num2str(volu));
```

NOTA: es importante no exagerar en los valores del parámetro *Gain* del bloque Volumen en Simulink, ya que es muy posible que vuestros parlantes o audífonos sufran daño innecesario.

Para cerrar un modelo desde Matlab usamos la función *close_system*. Para guardar los cambios usamos la función *save_system*.

3.15 Compartir datos entre dos GUIs.

Con este programa se pretende mostrar la forma de intercambiar información entre dos GUI distintas. Por ejemplo, en un programa de chat entre dos PCs a través de puerto serial, los valores de baudios, puerto, bits de datos, etc., se pueden ingresar en una GUI aparte, mejorando la presentación del programa. Para intercambiar datos entre dos GUI simplemente declaramos los datos como variables globales. Ejemplo: *global dato*. Para este ejemplo, en una GUI se ingresan dos valores que serán sumados en otra GUI. La GUI de ingreso de datos se ha guardado con el nombre *secuencia*.

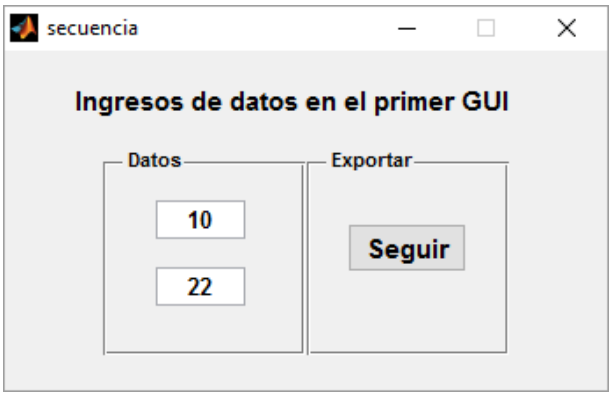

Figura 3.42: GUI de ingreso de datos.

Los campos *tag* de cada *edit text* son uno y dos. La programación del botón seguir (*tag*: SIGUIENTE) se muestra en el Listado 3.70.

Listado 3.70: Función del apertura.

```
global un
global do
un=str2double(get(handles.uno,'String'));
do=str2double(get(handles.dos,'String'));
close secuencia
seguir
```

El programa donde se suma estos valores se guarda con el nombre *seguir*.

En la parte del *m-file* donde se programan las condiciones iniciales del GUI editamos el código del Listado 3.71.

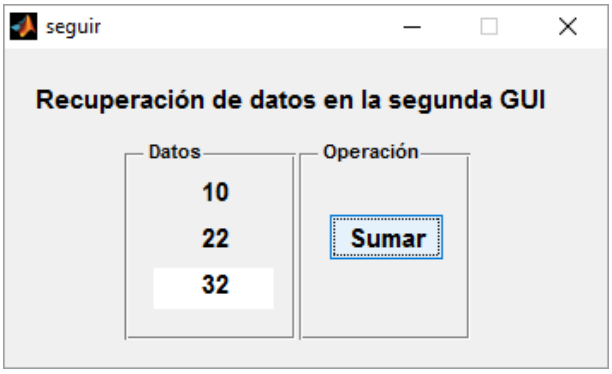

Figura 3.43: GUI donde se procesan los datos.

Listado 3.71: Función del apertura.

```
function seguir_OpeningFcn(hObject, eventdata, handles,
   varargin)
global un
global do
set(handles.text1,'String',un);
set(handles.text2,'String',do);
handles.output = hObject;
guidata(hObject, handles);
```

Con esto obtenemos los datos ingresados en el otro GUI. Para sumarlo, el código del botón *sumar* se muestra en el Listado 3.72.

Listado 3.72: Función del apertura.

```
function sumar_Callback(hObject, eventdata, handles)
global un
global do
s=un + do;
set(handles.text3,'String',s);
```

3.16 Captura de vídeo en GUI.

Con este programa se mostrará la forma de capturar vídeo e imágenes en una interfaz gráfica usando una web cam de PC.

El campo *tag* del axes que presenta el vídeo es *video_cam* y del axes que contiene la imagen capturada es *fotografia*. El código en la sección de inicialización del programa se muestra en el Listado 3.73.

Listado 3.73: Función del apertura.

```
function video_gui_OpeningFcn(hObject, eventdata, handles,
   varargin)
movegui(hObject,'center')
set(handles.axes1,'XTick',[],'YTick',[])
set(handles.axes2,'XTick',[],'YTick',[])
imaqreset
handles.output = hObject;
guidata(hObject, handles);
```

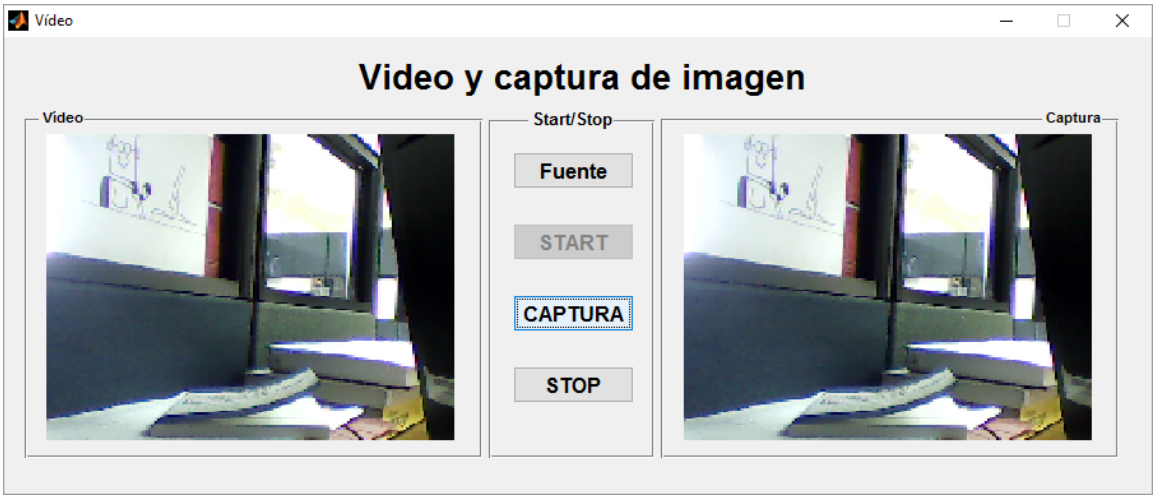

Figura 3.44: GUI de captura de vídeo e imagen.

Con la herramienta *Object browser* podemos ver el campo *tag* de cada elemento de la GUI. (Ver Figura 3.45).

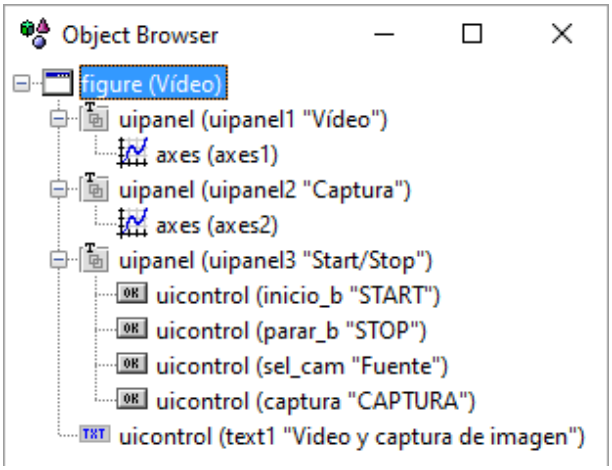

Figura 3.45: Interfaz del navegador de objetos.

El botón de inicio de la GUI abre un programa que de forma automática detecta las cámaras de vídeo (Ver Figura 3.46).

En la función de apertura contiene el código del Listado 3.74.

Listado 3.74: Función del apertura.

```
1 function sel_camera_OpeningFcn(hObject, eventdata, handles,
     varargin)
2 imaqreset;
3 set(handles.ok_b,'Enable','off')
4 hw=imaqhwinfo('winvideo');
5 handles.cam=hw;
6 set(handles.lista_camaras,'String',{hw.DeviceInfo.DeviceName
     })
7 handles.output = hObject;
8 guidata(hObject, handles);
```

Con el *pop up menu* seleccionamos qué cámara usar. Dentro de la función asociada a este elemento (mostrada en el Listado 3.75) está la programación que extrae las prin-

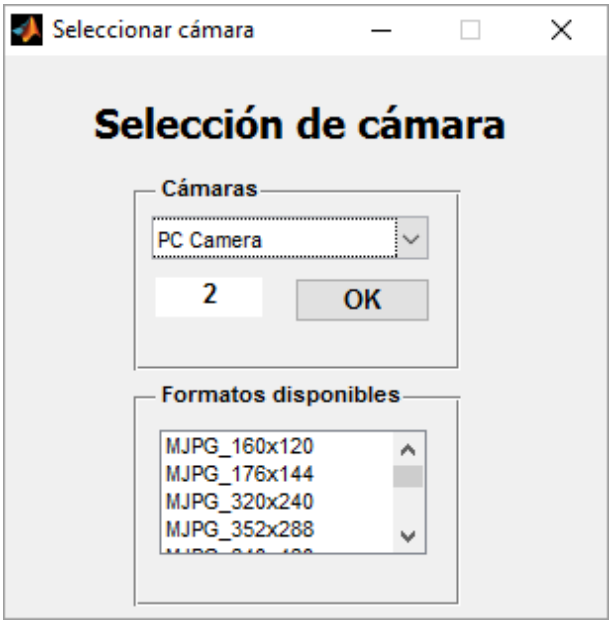

Figura 3.46: GUI de selección de cámara web.

cipales características de la cámara seleccionada.

Listado 3.75: Función del apertura.

```
function lista_camaras_Callback(hObject, eventdata, handles)
pos=get(handles.lista_camaras,'Value');
hw=handles.cam;
id=hw.DeviceIDs{pos};
set(handles.id_camara,'String',id)
formatos=hw.DeviceInfo(pos).SupportedFormats;
set(handles.formatos,'String',formatos)
list_f = [formatos{1:end}];
si=strfind(list_f,'RGB24_320x240');
if isempty(si)
    es_web_ext=0;% Laptop: YUY2
else
    es_web_ext=1;% External: RGB
end
handles.es_web_ext=es_web_ext;
handles.id=id;
guidata(hObject, handles)
set(handles.ok_b,'Enable','on')
```

Al presionar el botón *Ok* se exportan las características de la cámara seleccionada. El Listado 3.76 contiene el código de la función del botón. Note que se usan variables globales, al igual que el programa de la sección anterior.

Listado 3.76: Función del apertura.

```
function ok_b_Callback(hObject, eventdata, handles)
global id es_web_ext
es_web_ext = handles.es_web_ext;
id = handles.id;
close (handles.camara)
```

Regresando a la programación del la primera GUI, una vez seleccionada la fuente de vídeo, el botón de inicio muestra la secuencia de imágenes en el primer axes de la GUI (Ver Listado 3.77).

Listado 3.77: Función del apertura.

```
function inicio_b_Callback(hObject, eventdata, handles)
set(handles.inicio_b,'Enable','off')
start(handles.vidobj);
vidRes = get(handles.vidobj,'VideoResolution');
nBands = get(handles.vidobj,'NumberOfBands');
hImage = image(zeros(vidRes(2), vidRes(1), nBands),'Parent',
   handles.axes1);
preview(handles.vidobj,hImage);
guidata(hObject, handles);
```

Con el botón de captura, cuya programación se muestra en el Listado 3.78, obtenemos la imagen a partir del vídeo.

Listado 3.78: Función del apertura.

```
function captura_Callback(hObject, eventdata, handles)
try
    rgb = getsnapshot(handles.vidobj);
    if handles.es_web_ext==0
        rgb = ycbcr2rgb(rgb);
    end
    image(rgb,'Parent',handles.axes2);
    axes(handles.axes2)
    axis off
catch
    disp('No hay imagen para mostrar')
end
```

Finalmente, el botón de parada detiene la proyección de vídeo y permitirá seleccionar una nueva fuente de vídeo. (Ver Listado 3.79.)

Listado 3.79: Función del apertura.

```
function parar_b_Callback(hObject, eventdata, handles)
set(handles.inicio_b,'Enable','on')
stoppreview(handles.vidobj)
```

3.17 Tips de GUI.

Esta sección estará dedicada a exponer algunas características de las interfaces gráficas.

Editar dentro de un mismo string

set(handles.angulo,'string',['Angulo= ',num2str(ang),' Grados']);

Cambiar color de fondo de una interfaz

En la parte de las condiciones iniciales se coloca el código del Listado 3.80.

Listado 3.80: Función del apertura.

```
1 set(gcf,'Color',[0.5 0.9 0.5])
```

El vector *[0.5 0.9 0.5]* contiene los valores RGB (red, green, blue). Cada uno de los elementos de esta matriz va de 0 hasta 1.

Escribir varias líneas en un edit text

Simplemente se coloca los valores máximos y mínimo como 5 y 3 respectivamente.

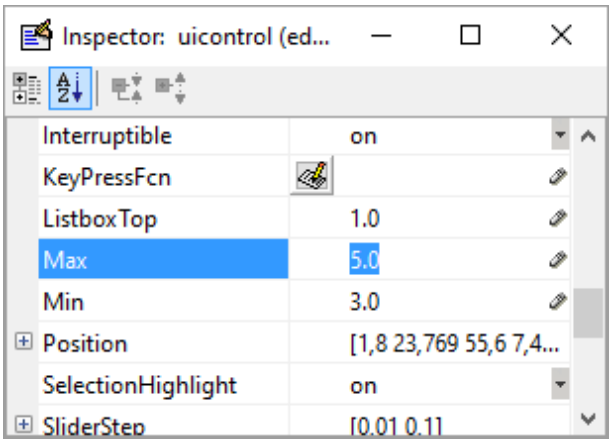

Figura 3.47: Variar valor máximo y mínimo de las propiedades.

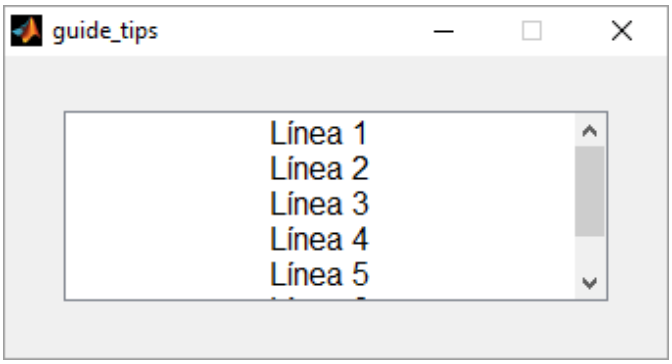

Figura 3.48: Ventana de edición de texto con múltiples líneas.

Colocar la interfaz gráfica en el centro de la pantalla

Para conseguir que la interfaz aparezca en el centro de la pantalla, colocamos el código del Listado 3.81 en la función de condiciones iniciales del programa.

Listado 3.81: Función del apertura.

```
1 scrsz = get(0, 'ScreenSize');
2 pos_act=get(gcf,'Position');
3 xr=scrsz(3) - pos_act(3);
4 xp=round(xr/2);
5 yr=scrsz(4) - pos_act(4);
6 yp=round(yr/2);
7 set(gcf,'Position',[xp yp pos_act(3) pos_act(4)]);
```

Las unidades de la interfaz gráfica deben estar en píxeles.

Cambiar el tamaño de la interfaz

Haciendo doble clic en el área de diseño de la interfaz, activar el parámetro *resize*.

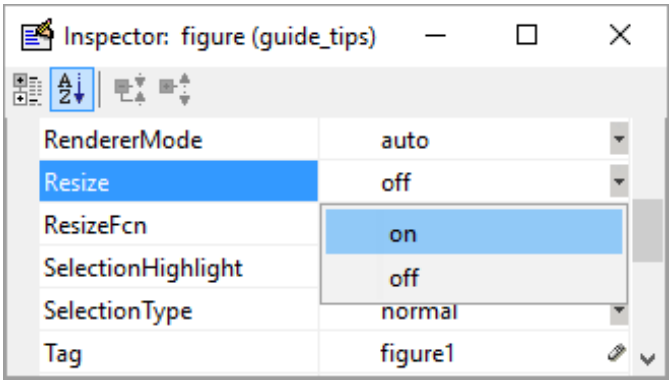

Figura 3.49: Redimensionar la GUI.

Cambiar el cursor de la interfaz

Haciendo doble clic en el área de diseño de la interfaz, cambiar el parámetro *pointer*.

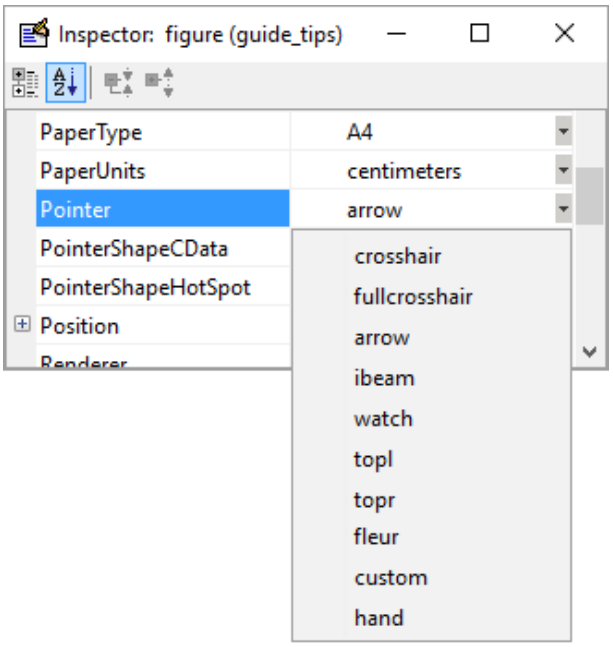

Figura 3.50: Cambio de estilo del cursor de la GUI.

Cambiar de nombre a una interfaz

Aunque esto parezca sencillo, cuando se cambia el nombre a una GUI con *rename* se produce un error. Esto es consecuencia de que el nombre de la interfaz debe ser el mismo de la función principal.

Para cambiar el nombre a la interfaz simplemente se usa *Save As* en el archivo .fig.

Obtener el código ASCII usando KeyPressFcn

Luego de crear la función KeyPress, añadir la siguiente función:

Listado 3.82: Función del apertura.

```
1 a=double(get(gcbf,'Currentcharacter'));
```

Con esta función se captura el código ASCII en la variable a.

Eliminar código excesivo

Por lo general las funciones *CreateFcn* no se usan en una interfaz de usuario (*CreateFcn* define las condiciones iniciales de un objeto, pero las podemos establecer en la función de apertura del programa). Para que no se genere el código correspondiente a estas funciones, simplemente antes de guardar por primera vez la interfaz, en la parte de *CreateFcn* eliminar la palabra *automatic* y el código no se generará. Ver Figura 3.51.

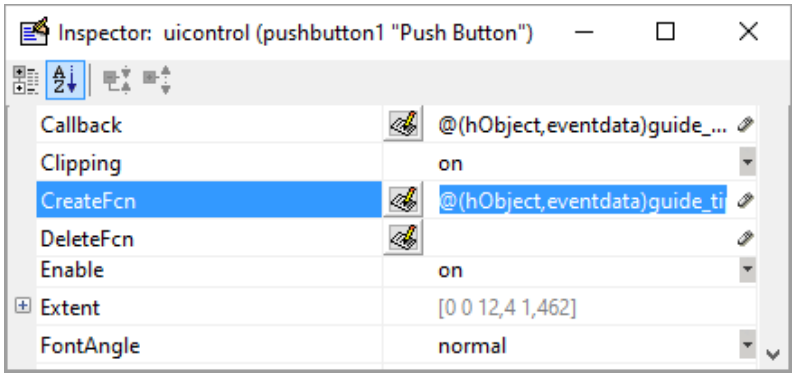

Figura 3.51: Eliminar código de apertura de la función.

Arreglar el orden del TAB

Para ordenar los saltos de la tecla *tab* en la GUI, se usa la herramienta *Tab Order Editor*, mostrada en la Figura 3.52.

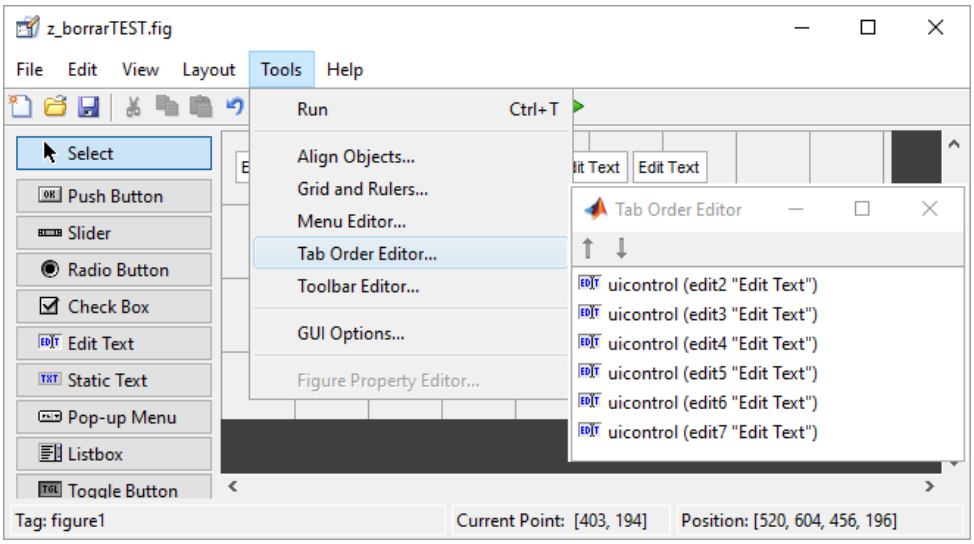

Figura 3.52: Orden de la tecla *tab*.

Anexo A

Autor

Diego Barragán Guerrero (FiguraA.1) es profesor universitario formado por la UTPL y la UNICAMP. Entre sus intereses están el Tratamiento Digital de Señales, Comunicaciones Digitales y Sistemas Embarcados (VHDL).
Por informarme de cualquier error en el texto, quedo muy agradecido.

Figura A.1: Ping pong is more fun when you win.

8 November 2016