TEMA 4

Interficie gráfica con el usuario

1. Introducción

El sistema gráfico de MATLAB permite:

- Visualizar datos mediante comandos de alto nivel. Esto incluye la visualización en 2 y 3 dimensiones, el procesado de imágenes/fotos, la elaboración de gráficos

para presentaciones (diagramas de barras, de queso) y la inclusión de efectos (de animación, iluminación, movimientos de cámara).

- Crear y manipular objetos gráficos mediante comandos de bajo nivel. Se trata de las herramientas del GUI (*Graphics User Interface*) que permiten la programación de aplicaciones orientadas al usuario, con ventanas, menús y controles. Estas herramientas son las que se verán en el presente tema.

2. *Graphics User Interface* (GUI)

El GUI (*Graphics User Interface*) de MATLAB es el conjunto de herramientas y comandos de bajo nivel que permiten crear y manipular objetos gráficos con el fin de desarrollar aplicaciones "amigables" para el usuario.

Los objetos gráficos de MATLAB presentan la siguiente jerarquía:

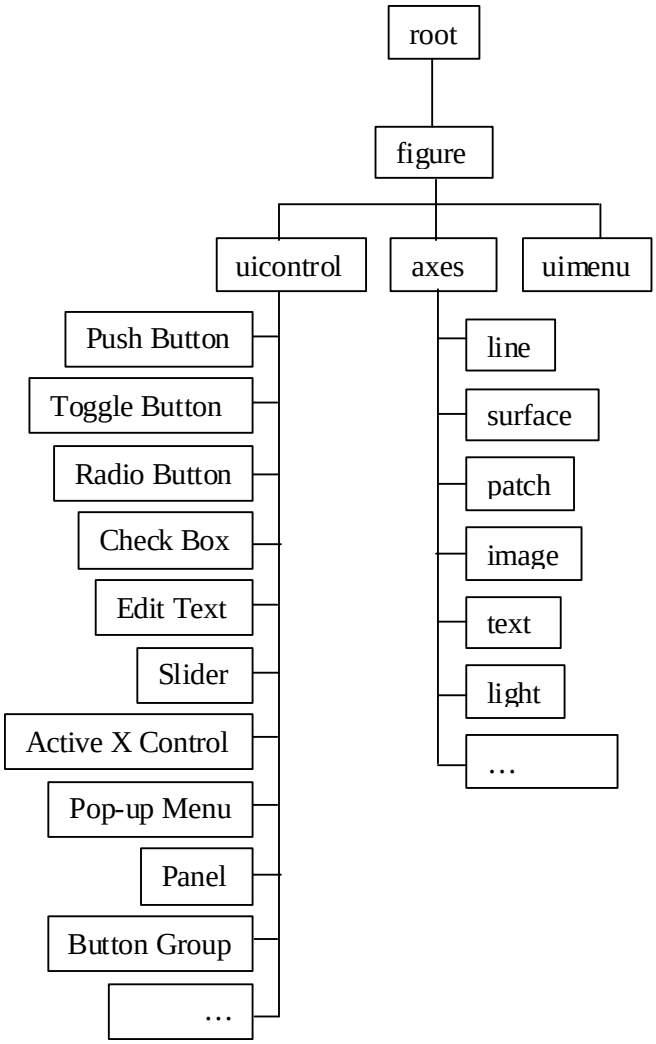


Fig. 1. Jerarquía de los objetos gráficos en MATLAB.

Objetos: El objeto **`root`** es la pantalla principal. Ningún objeto de los anteriores puede existir si no existe **`root`** (es decir, si no está abierto el programa MATLAB no podemos tener ni **`figure`**'s, ni **`axes`**, etc.).

Cuando en **`root`** hacemos **`>>plot(t,y)`** se crean todos los objetos necesarios para la representación gráfica de *y* en función de *t* (**`figure`** → **`axes`** → **`line`**), si es que antes no existían.

Handle y propiedades: Todos los objetos gráficos tienen un *handle* (número que lo identifica) y un conjunto de propiedades (**`Color`**, **`Position`**, etc.). Algunas de las propiedades pueden modificarse pero otras no. El *handle* del objeto **`root`** es 0.

Las propiedades de los objetos son accesibles desde los comandos de texto de bajo nivel (**`set`**, **`get`**) y desde las herramientas del editor gráfico **`guide`** (*Property Inspector*, *Object Browser*, *Menu Editor*).

3. Comandos de bajo nivel: *set* y *get*

Los dos comandos de texto de bajo nivel son **`get`** y **`set`**. El primero devuelve el valor de una, varias o todas las propiedades de un objeto gráfico y el segundo las modifica (si es posible).

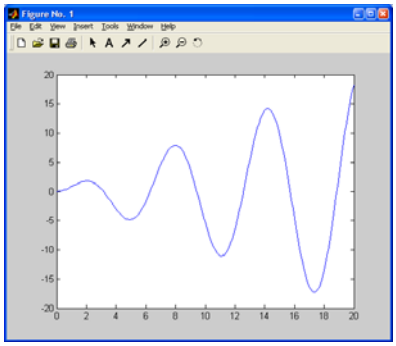
La función **`set`** también puede utilizarse para ver qué propiedades son modificables y cuáles son los valores válidos. Por ejemplo, ver qué sucede al teclear:

```
>> set(text)
```

Ejemplo 1. Objetos, identificadores (*handles*) y propiedades

1) Generar un vector temporal de 0 a 20*s* en pasos de 0.1*s* y representar la curva $y = t \cdot \sin(t)$ con ayuda de la función **`plot`**.

```
» t=0:0.1:20;y=t.*sin(t);plot(t,y)
```



2) Obtener el identificador del objeto figure actual.

```
» gcf
ans =
     1
```

Nota: **`gcf`** significa ***G**et handle to **C**urrent **F**igure*

Nota: **`gca`** hace lo mismo que **`gcf`** pero para el objeto **`axes`** y **`gco`** es para cualquier tipo de objeto (hay que seleccionarlo antes).

```
» gca
ans =
```

```
  102.0009

» gco  % antes hay que haber clicado en el objeto line de la fig
ans =
    4.0002
```

Nota: clicar ahora en los ejes y ver que efectivamente **gco** devuelve el handle correspondiente al objecto *axes*.

```
» gco
ans =
  102.0009
```

3) Obtener las propiedades 'Color' y 'Position' de dicha figura:

```
» get(gcf,'color')
ans =
    0.8000    0.8000    0.8000

» get(gcf,'position')
ans =
   232   258   560   420
```

Nota: los nombres de las propiedades se pueden abreviar (si no existe posibilidad de confusión). Así, también se pueden utilizar, entre otras, las siguientes expresiones:

```
» get(gcf,'pos')
» get(gcf,'Posi')
```

4) Modificar la propiedad 'Position'. Determinar a qué se refiere cada una de las cuatro componentes del vector

```
» set(gcf,'pos',[608 95 110 300])
```

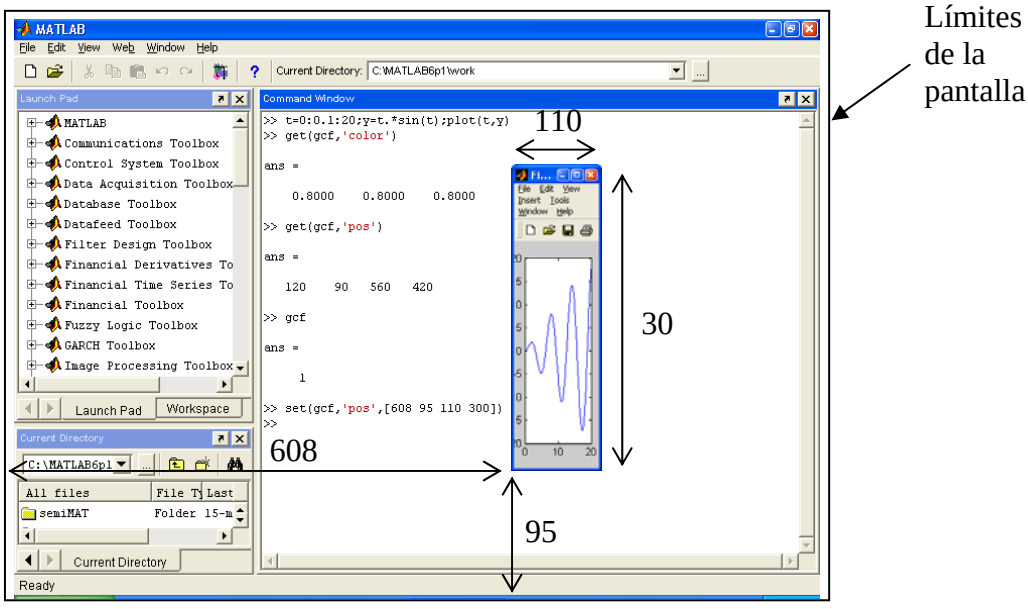


Fig. 2. Propiedad "Position"

5) Ver todas las propiedades de la figura actual con ayuda de **get** ¿De qué objeto es padre (ver propiedad 'Children')? ¿De qué objeto desciende (ver propiedad 'Parent'?:

```
» get(gcf)
```

6) Ver todas las propiedades que es posible modificar con ayuda de **set** ¿Qué propiedades no es posible modificar?

```
» set(gcf)
```

7) Cerrar todas las figuras:

```
» close all
```

y verificar que hacer

```
» line
```

es equivalente a hacer

```
» figure
» axes
» line
```

Notar lo que hace cada instrucción y cómo va la numeración de los objetos.

Sugerencia: Repetir para el resto de objetos gráficos, hacer por ejemplo:

```
» uicontrol
» patch
» surface
```

etc.

4. Editor de propiedades

4.1 Editor de propiedades en la versión 6.x

Ejemplo 2. Editor de propiedades en la v6

Volver a representar la curva $y = t \cdot \sin(t)$ en función de un vector temporal de 0 a 20*s* en pasos de 0.1*s* y ajustar de nuevo la propiedad **`Position`** del objeto **`figure`** que contiene la representación.

```
>> t=0:0.1:20;y=t.*sin(t);plot(t,y)
>> set(gcf,'pos',[608 95 110 300])
```

Property Editor: Para entrar en el editor de propiedades seleccionar, en la propia figura, la opción **`Edit → Figure Properties...`** Notar que el editor tiene dos áreas:

en la superior hay una lista de los objetos gráficos existentes (ordenados jerárquicamente) y, en la inferior, están las propiedades asociadas al objeto seleccionado en la lista anterior. Notar también que, al seleccionar un objeto desde el editor, éste aparece seleccionado en la figura.

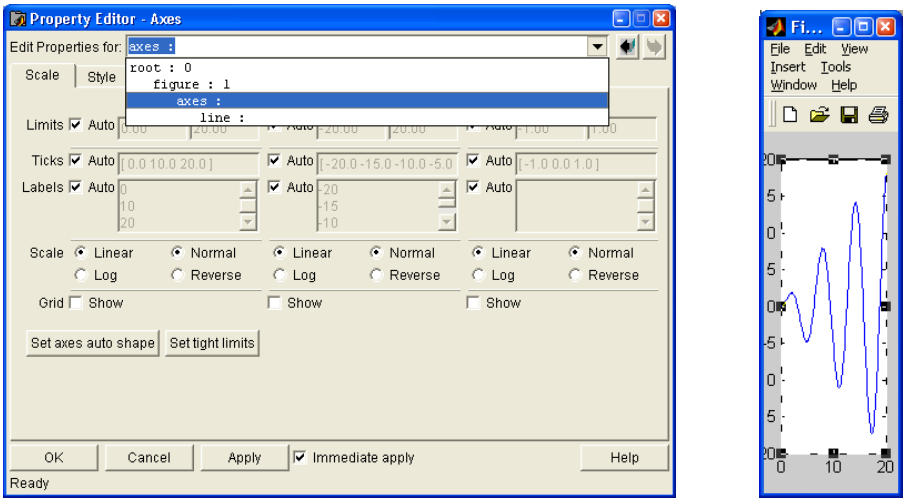


Fig. 3. Editor de propiedades en la versión 6.1.

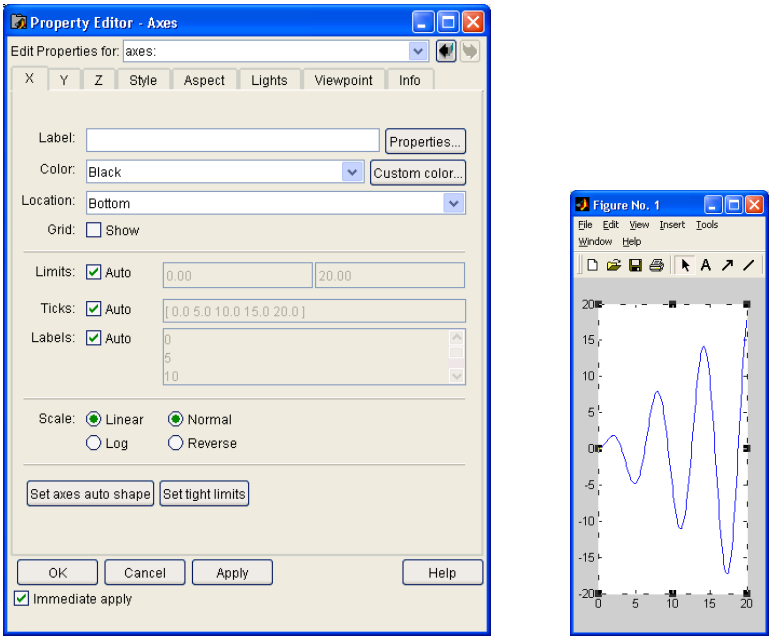


Fig. 4. Editor de propiedades en la versión 6.5.

4.2 Editor de propiedades en la versión 7.x

Ejemplo 3. Editor de propiedades en la v7

Volver a representar la curva $y = t \cdot \sin(t)$ en función de un vector temporal de 0 a 20s en pasos de 0.1s.

```
>> t=0:0.1:20;y=t.*sin(t);plot(t,y)
```

Property Editor: Para entrar en el editor de propiedades seleccionar, en la propia figura, la opción **Edit → Figure Properties...** o bien **View → Property editor**. Notar que si se selecciona uno de los objetos, aparece el editor de propiedades correspondiente al objeto seleccionado:

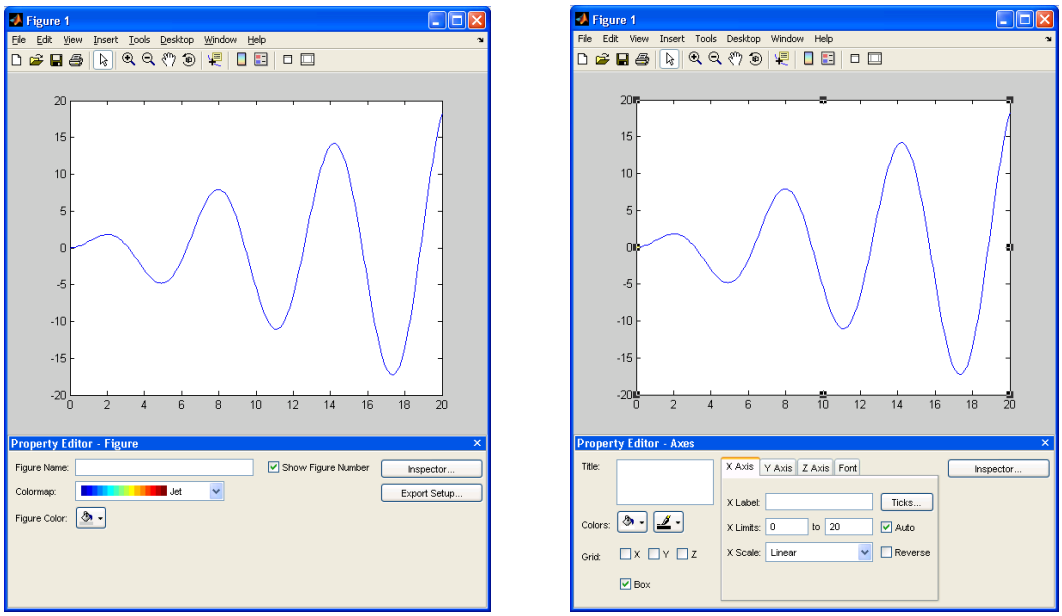
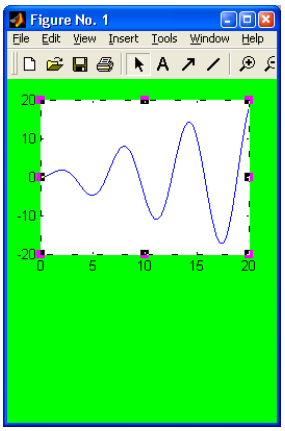


Fig. 5. Editor de propiedades en la versión 7.0.4.

4.3 Modificación de objetos con el editor de propiedades

Ejemplo 4. Modificación de propiedades v6 y v7

Versión 6.x: Modificar el color de fondo del objeto **figure** a verde (**Green** o [0 1 0]: recordar que el color es la terna con los porcentajes de RGB) y cambiar también la propiedad **Position**. Para modificar el objeto **axes**, seleccionarlo en la lista de objetos y escribir las nuevas coordenadas (también es posible, y más cómodo, ajustar los ejes con ayuda del ratón: clicar con el derecho sobre los ejes, seleccionar la opción **Unlock Axes Position** y arrastrar).



Versión 7.x: Modificar el color de fondo del objeto **figure** a verde (para ello usar el botón "Figure Color"). Asimismo, modificar el tamaño del objeto **figure** con ayuda del ratón. Para modificar el objeto **axes**, clicar sobre él y arrastrar su contorno con el ratón a fin de hacerlo más pequeño.

Ambas versiones: Salvar la figura (**File→Save**) como **ejemplo4.fig** y cerrarla. Para volver a abrir la figura se puede hacer:

```
>>openfig('ejemplo4')
```

---

5. Panel de controles. Utilidad GUIDE

Clicar en  o bien desde la ventana de comandos hacer >>**guide** y, desde ahí, abrir el fichero creado en el ejemplo anterior **ejemplo4.fig**. También es posible hacer >>**guide('ejemplo4')**. (Nota: Si en la versión 7 aparece un cuadro de **axes** con la leyenda *scribeOverlay*, clicar sobre él y borrarlo).

Paleta de controles: Por defecto, en la librería de objetos sólo aparecen los iconos. Si se quiere ver el nombre completo de cada objeto ir a **File→Preferences…** y seleccionar la opción **Show names in component palette**.

5.1 Creación y edición de objetos desde la utilidad GUIDE

Creación y alineación de objetos: Arrastrar dos botones (**push button**) a la figura y, opcionalmente, alinearlos a partir de la opción de menú **Tools→Align Objects…** o a partir del icono correspondiente  :

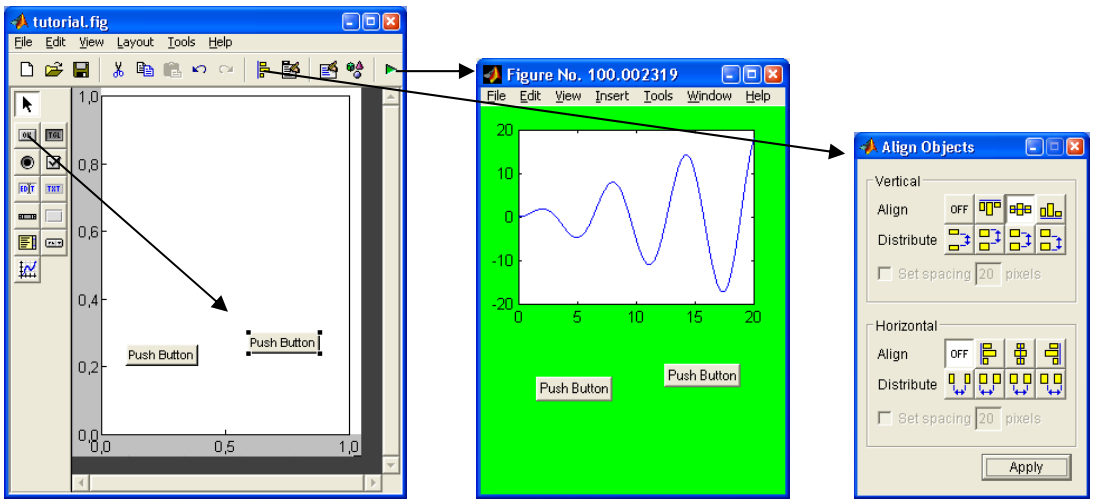


Fig. 6. Alineación de objetos (v6.x)

5.2 Object browser. Propiedades Tag y String

Para abrir el *Object Browser* se puede hacer desde el menú **`View→Object Browser`** o bien clicando en el icono  . En el *Object Browser* aparece la lista de todos los objetos creados en la GUI (ver Fig. 7).

Propiedades Tag y String: En concreto, en el *Object Browser* aparecen entre paréntesis las propiedades **`Tag`** y **`String`** de cada objeto.

El **`Tag`** es un alias (nombre) que permite referenciar los objetos sin tener que recurrir al número de *handle*. En la Fig. 7 los **`Tag`** de los dos botones son **`pushbutton1`** y **`pushbutton2`**. Éstos son los nombres que GUIDE les da por defecto, lo habitual es que el usuario cambie estos Tags y los sustituya por otros nombres más relacionados con lo que hacen los controles al activarse.

La propiedad **`String`** hace referencia al texto que aparece sobre cada control (notar que se muestra entre comillas). En la Fig. 7 el nombre (**`String`**) que aparece sobre cada botón es **`Push Button`**. Éste también es el **`String`** por defecto para los botones.

Para acceder a la totalidad de las propiedades de un objeto se puede hacer clicando sobre el objeto o clicando sobre su nombre en el *Object Browser*.

5.3 Property Inspector. Modificación de propiedades

Para abrir el *Property Inspector* se puede hacer desde la opción del menú **`View→Property Inspector`**, o clicando sobre el icono  , o haciendo doble clic en el objeto en cuestión, o clicando sobre el nombre del objetos en el *Object Browser*. En el *Property Inspector* aparece la lista de todas las propiedades del objeto así cómo los valores asignados a estas propiedades (ver Fig. 7).

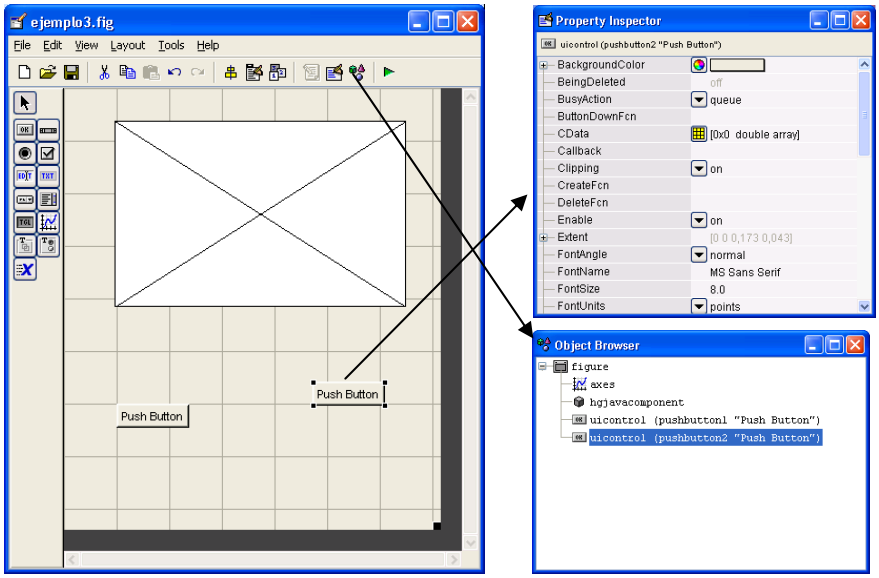


Fig. 7. Object Browser y Property Inspector (v7.x)

Modificación de las propiedades de un objeto (String y Callback): Seleccionar, con el ratón o desde el *Object Browser*, uno de los botones.

Hacer doble clic sobre el botón seleccionado para abrir el *Property Inspector* Cambiar las propiedades **`String`** y **`CallBack`** escribiendo en ambas **`grid on`**, pulsar <**`enter`**> y cerrar. Ídem con el otro botón pero escribir **`grid off`**.

La propiedad **`String`** hace referencia al texto que aparecerá sobre el botón. La propiedad **`Callback`** hace referencia al comando o comandos de usuario que se ejecutarán al ser pulsado el botón.

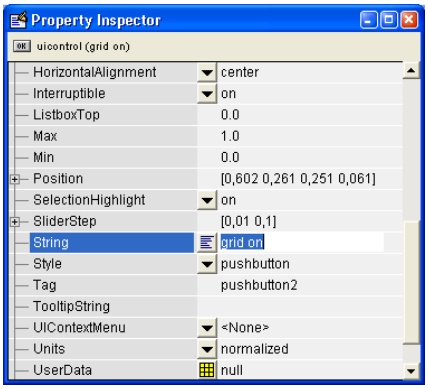


Fig. 8. Modificación de propiedades

Activar la Figura clicando en  o bien desde el menú (**`Tools→Activate Figure`**) o (**`Tools→Run`**, según la versión). Una vez activada ya es posible ejecutar los controles:

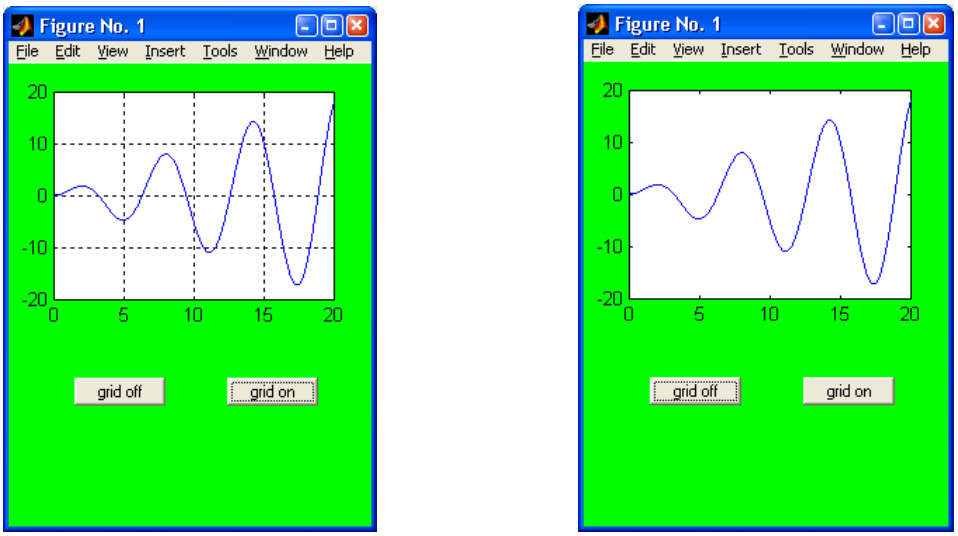


Fig. 9. Verificación del comportamiento de la GUI una vez activada

Propiedad NextPlot: A veces, al crear un objeto gráfico (por ejemplo, unos **`axes`**) es deseable cambiar las propiedades por defecto y poner otras (cambiar el color, los límites de los ejes, poner una rejilla,…). Estas propiedades del objeto **`axes`** se pueden perder en situaciones donde, como respuesta a la activación de otros controles, en estos **`axes`** se va cambiando el objeto hijo **`line`** representado (por ejemplo, al hacer clic en un botón se representa un seno y al volver a hacer clic un coseno). Para evitar esto hay que modificar la propiedad **`NextPlot`**. Los valores posibles son:

- **`New`**: se crea un objeto (**`figure`** o **`axes`**) nuevo cada vez

- **`Add`**: se usa el objeto tal como está
- **`Replace`**: se eliminan todos los objetos hijos (**`line`**,…) anteriores y se resetean todas las propiedades excepto la posición
- **`Replacechildren`**: se eliminan los objetos hijos pero no se resetean las propiedades

Otros comandos relacionados con la limpieza de la pantalla son **`nextplot`** y **`drawnow`**.

5.4 Menu Editor. Función findobj

Para abrir el *Menu Editor* puede hacerse desde la opción de menú **`Tools→Menu Editor…`** o bien clicando en el icono  .

Siguiendo con el ejemplo vamos a crear un nuevo menú de nombre "grids" con dos opciones (ítems) "grid on" y "grid off" que hagan lo mismo que los botones anteriores.

Para crear un nuevo menú clicar en  . Asignar el valor "grids" a la propiedad **`Label`** y a la propiedad **`Tag`**. **`Label`** es el nombre que saldrá en el menú.

Crear dos submenús (clicar en ) dependientes del menú "grid" con las propiedades Label y Callbacks **`grid on`** y **`grid off`**. Notar que, para establecer qué menús cuelgan de cuáles, se pueden utilizar los símbolos (→, ←, ↓, ↑).

Una vez editados los menús, cerrar el editor, activar la figura y verificar su comportamiento.

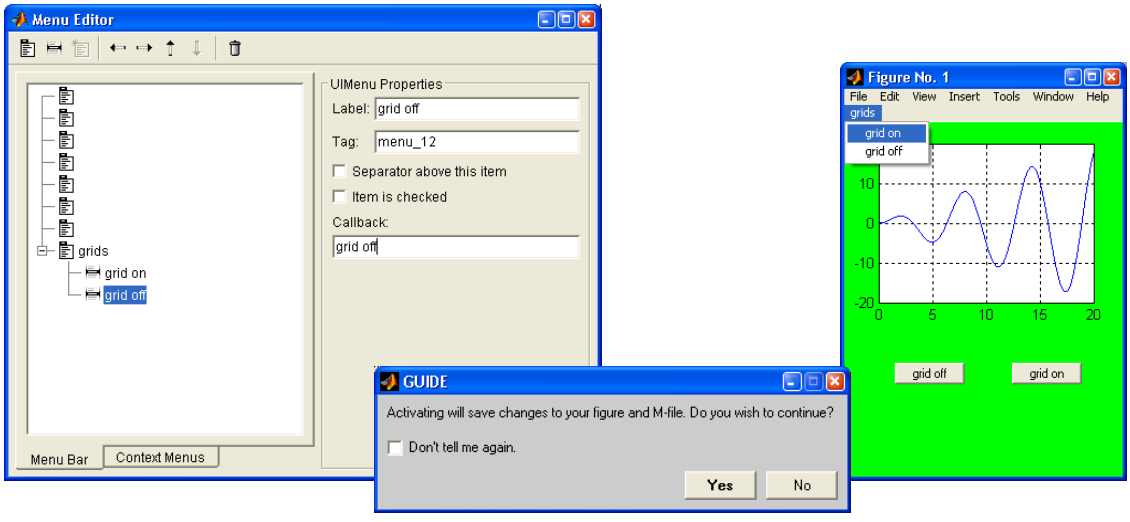


Fig. 10. Edición de menús (v6.x)

Función findobj: Una función útil en el desarrollo de aplicaciones es la función **`findobj`**, que devuelve el *handle* de todos los objetos cuya propiedad tiene un determinado valor. Por ejemplo, si tenemos varias opciones de menú y queremos saber cuál está seleccionada podemos hacer:

```
h=findobj('Checked','on');
```

Editor de propiedades: Notar que los nuevos objetos (controles y menús) ya serán modificables desde la Figura (**`Edit→Figure Properties`**) sin necesidad de volver a abrir el **`guide`**:

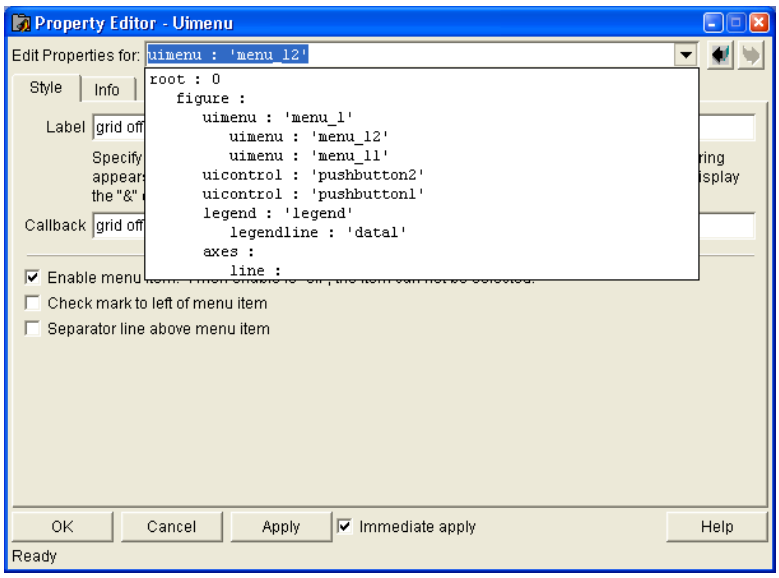


Fig. 11. Editor de propiedades en v6.x

Finalmente, si ya se tiene una figura abierta y se quiere abrir el **`guide`** para editarla o añadirle objetos, es posible hacer `>>guide(gcf)`.

6. Desarrollo de aplicaciones

Los pasos para el diseño de aplicaciones son los siguientes:

1. Hacer un croquis de la apariencia que debe tener la GUI y su funcionalidad.

2. Crear la ventana principal y arrastrar a ella todos los controles. Alinear, distribuir y ordenar los controles.

3. Establecer las propiedades de los controles. Algunas de las propiedades típicas que se establecen en esta etapa del diseño son: **`String`**; **`Tag`**; **`Color`**; en cuadros de texto: **`BackgroundColor`**, **`ForegroundColor`**, **`FontSize`**...; en paneles: **`Title`**; en figures: **`Name`**; en axes: **`XLim`**, **`YLim`**,...

   La propiedad **`Callback`** es la propiedad que usa el programador para que la GUI tenga el comportamiento deseado. Si la tarea que se tiene que ejecutar al activar un control es simple, lo normal es poner esta tarea en el campo **`Callback`** del *Property Inspector*. Por ejemplo, **`Callback=grid on`** o **`Callback=close`**.

Pero cuando la tarea es compleja (realizar unos cálculos y representarlos con cierta apariencia, por ejemplo) es mejor usar el fichero *.m que se crea por defecto al salvar el fichero *.fig o bien usar otro fichero *.m escrito por el usuario (esta segunda opción se verá a continuación, en el apartado 7, para ver un ejemplo donde se edita el fichero *.m que se crea por defecto, ver el apartado 8).

4. Crear los menús y submenús y asignarles las propiedades **`Label`**, **`Tag`** y **`Callback`** (si éste es simple).

5. Editar el fichero *.m que controla el comportamiento de la aplicación. Se recomienda ir verificando el comportamiento de cada control a medida que se edita este fichero.

6. Verificar el comportamiento de la GUI en su totalidad.

7. Ejemplo (versión 6.x)

7.1 Creación de la ventana principal y de los controles

Guide: Para abrir la utilidad GUIDE teclear **`>>guide`** en la ventana de comandos o bien clicar en el icono  de la barra de herramientas.

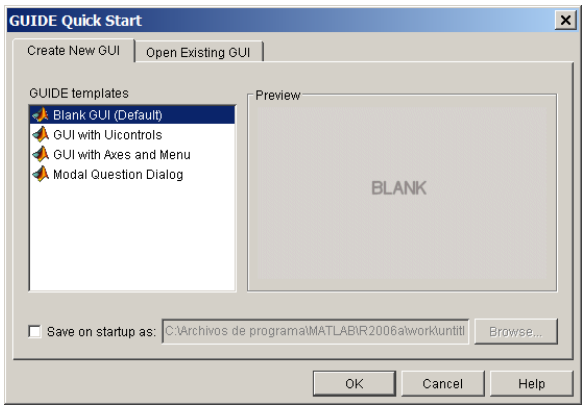


Fig. 12. GUIDE

Figura principal: Seleccionar la opción **`Blank GUI (Default)`** a fin de crear una **`figure`** nueva. En el menú de la figura, seleccionando "*Tools→GUI Options*" se puede cambiar la propiedad de redimensionado (el valor por defecto es *Non-resizable* pero se puede ajustar a *Proportional*).

Creación de los objetos: Insertar los objetos que se muestran en la Fig. 13 (dos **`axes`**  , dos **`push buttons`**  , un **`edit text`**  y un **`static text`** ).

Propiedades de los objetos: Abrir el **`Property Inspector`**  de los siguientes objetos y establecer las propiedades indicadas:

Objeto	**`String`**	**`Tag`**	**`Callback`**
Edit text	frec	frec	tema4('refresh')
Static text	Frecuencia (rad/s)		
Push button 1	grid		grid
Push button 2	SALIR	SALIR	tema4('salir')

Verificar que en el *Object Browser* aparecen los Strings (entre comillas) y los Tags de la tabla anterior. En cuanto a la propiedad Callback, esta propiedad permite que se ejecute una función Matlab cuando se activa el objeto. En nuestro ejemplo, al clicar en el **`pushbutton2`**, se ejecutará la subrutina **`'salir'`** de la función **`'tema4.m'`** (está función está definida en el siguiente apartado).

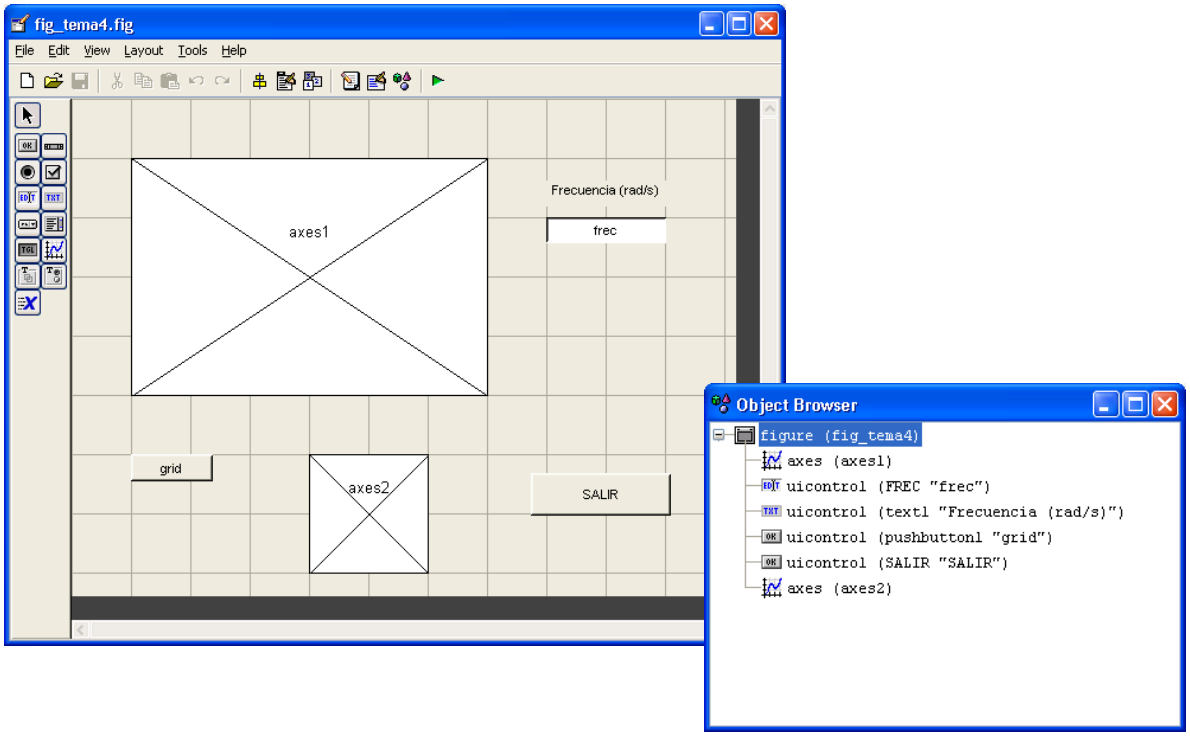


Fig. 13. Objetos en una GUI

Salvar la GUI con el nombre **`fig_tema4.fig`**.

7.2 Edición de la función *.m de la aplicación. Callbacks

Abrir el editor de ficheros M y editar la función siguiente (**`tema4.m`**)

```
function tema4(accion)

if nargin<1
    accion='ini';
end
```

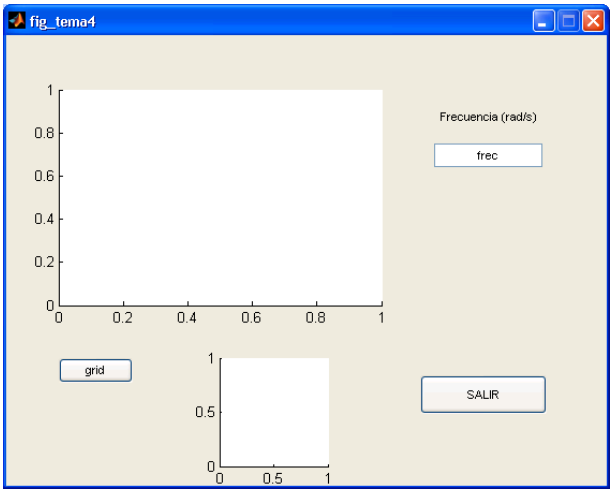
```
global ctrl

if strcmp(accion,'ini')

    f=openfig('fig_tema4');
    ctrl=guihandles(f)

elseif strcmp(accion,'refresh')

    frec=str2double(get(ctrl.FREC,'string'))
    t=0:0.1:20;
    y=t.*sin(frec*t);
    plot(t,y,'Parent',ctrl.axes1)

elseif strcmp(accion,'salir')

    close
    clear global ctrl

end
```

Notar la conversión **`str2double`** del valor **`frec`** y cómo se le ha indicado al objeto **`line`** en cuál de los dos **`axes`** debe situarse.

7.3 Ejecución de la aplicación

Basta con teclear el nombre del fichero *.m en la ventana de comandos. El resultado se muestra a continuación:

```
>> tema4

ctrl =

        figure1: 168.0074
          axes2: 14.0079
          SALIR: 13.0079
    pushbutton1: 12.0079
          text1: 11.0082
           FREC: 174.0072
          axes1: 169.0073
```



Notar que la función **`guihandles`** devuelve una estructura con los *handles* de los objetos de la GUI. Los campos de la estructura son los *tags* de los objetos.

En el cuadro *Edit* escribir diferentes valores de frecuencia y ver qué sucede al pulsar *Enter*. Probar el funcionamiento de los botones “grid” y “SALIR”.

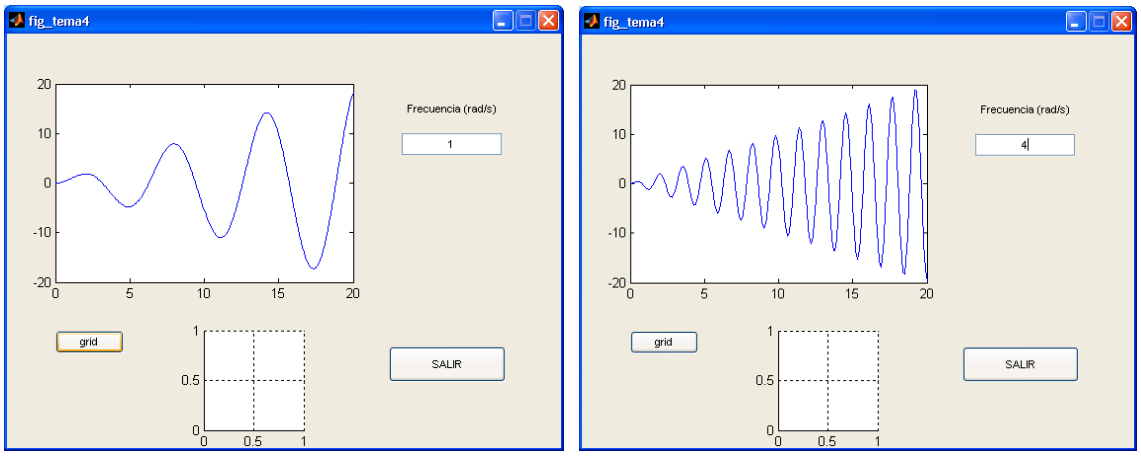


Fig. 14. Funcionamiento de la GUI

8. Ejemplo (versión 7.x)

8.1 Creación de la ventana principal y de los controles

Guide: Para abrir la utilidad GUIDE teclear **`>>guide`** en la ventana de comandos o bien clicar en el icono  de la barra de herramientas.

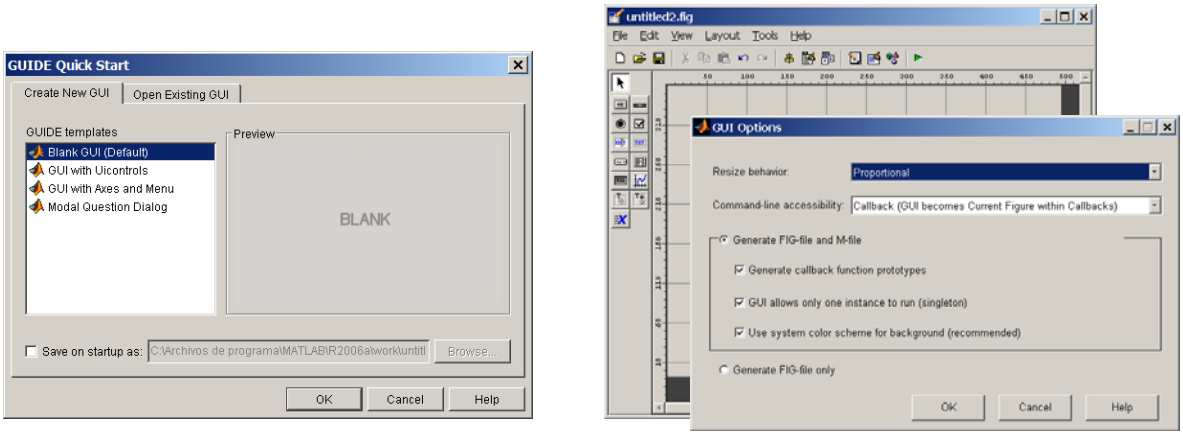


Fig. 15. GUIDE

Figura principal: Seleccionar la opción **`Blank GUI (Default)`** a fin de crear una **`figure`** nueva. En el menú de la figura, seleccionando "*Tools→GUI Options*" se puede cambiar la propiedad de redimensionado (el valor por defecto es *Non-resizable* pero se puede ajustar a *Proportional*).

Creación de los objetos: Insertar diversos controles (por ejemplo, un par de **`axes`**  , un **`pop-up menú`**  , un **`panel`**  , dos **`pushbuttons`** ). Poner los **`pushbuttons`** dentro del **`panel`** y observar cómo a partir de ahora se convierten en un grupo que se puede desplazar y redimensionar a la vez. En el caso de arrastrar y clicar en un control

ActiveX aparecerá una ventana listando todos los controles **ActiveX** disponibles en el ordenador.

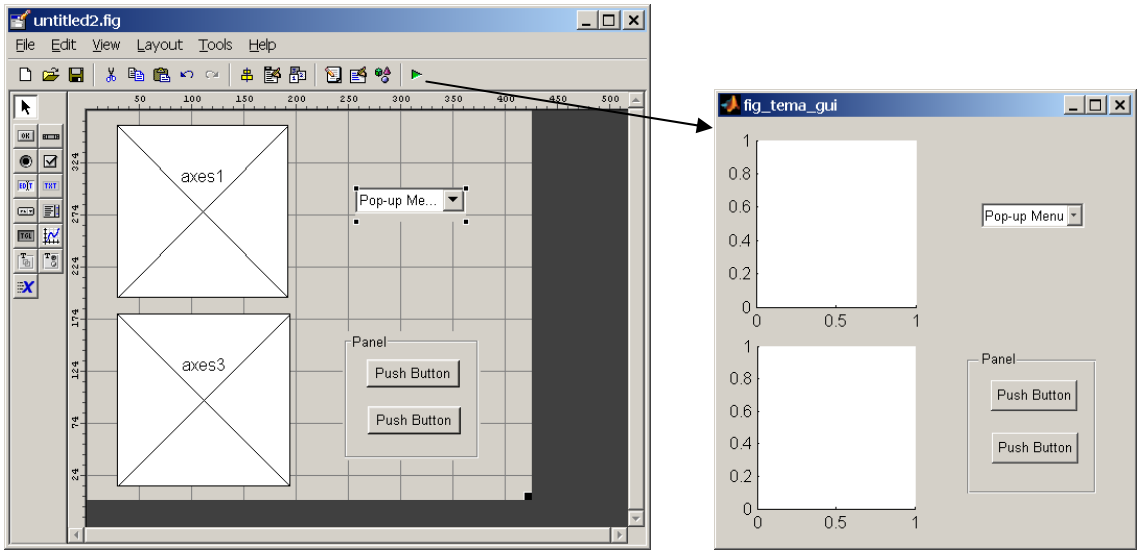


Fig. 16. Controles

Clicar en y para acceder a las herramientas de alineación, distribución y ordenamiento de objetos.

Propiedades de los objetos: Abrir el **Property Inspector** de los siguientes objetos y establecer las propiedades indicadas:

Objeto	**String**	**Title**	**Tag**	**Callback**
Push button 1	grid		bot_grid	
Push button 2	salir		bot_salir	close
Uipanel 1		botones		
Pop up menu 1	inicio|bolas|sombrero			

Por ejemplo, el *Property Inspector* del segundo **pushbutton** quedará así:

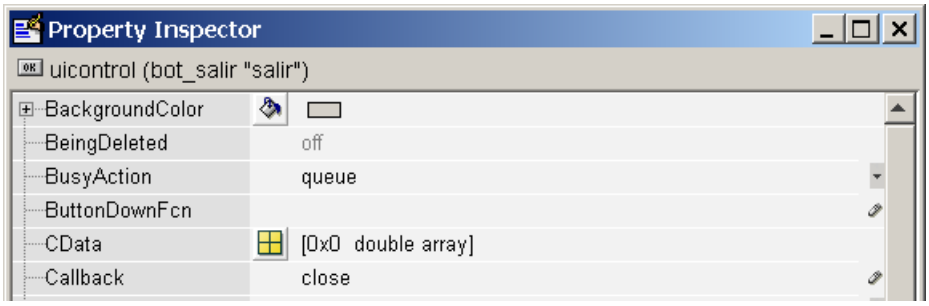


Fig. 17. Edición de las propiedades de los controles con el *Property Inspector*

Para cambiar el **String** del **popupmenu** clicar en el icono dentro de su *Property Inspector* y poner la lista de etiquetas en una columna.

Activación de la ventana y creación del fichero m: Clicar en  o seleccionar "**Tools→Run**". Indicar cuál será el nombre de la ventana, por ejemplo, "**demo_gui**". Por defecto se crean dos ficheros: **demo_gui.fig** y **demo_gui.m**. El fichero *.m asociado a la GUI puede abrirse desde el GUIDE, basta con clicar en  o hacer **View→M-file Editor**.

El contenido del fichero *.m se estructura en diversas funciones. Para verlas clicar en el icono  disponible en el editor de ficheros M.

El fichero *.m creado por defecto para nuestro ejemplo es el siguiente (versión 7.2):

```
function varargout = demo_gui(varargin)
% DEMO_GUI M-file for demo_gui.fig
%      Comentarios de ayuda

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @demo_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @demo_gui_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 demo_gui is made visible.
function demo_gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to demo_gui (see VARARGIN)

% Choose default command line output for demo_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes demo_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = demo_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- 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)


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Notar que hay una función de inicialización, una de finalización (funciones **`demo_gui_OpeningFcn`** y **`demo_gui_OutputFcn`**) y, a continuación, varias otras funciones, cada una correspondiente a un **`Callback`** de un objeto concreto.

Nota: Hay diferentes tipos de Callbacks, los Callbacks propiamente dichos y otros que se generan dependiendo del tipo de objeto (como, por ejemplo, CreateFcn, DeleteFcn, ButtonDownFcn, etc…). Para ver la lista entera de Callbacks se puede ir al guide y seleccionar la opción de menú **`View→View Callbacks`**. No es necesario rellenar todos los Callbacks de un objeto (a veces ya vienen rellenos por defecto con instrucciones de inicialización). Lo habitual es trabajar sólo con las propiedades Callback propiamente dichas.

En una aplicación típica, el usuario puede crear sus propias funciones auxiliares y poner las a continuación de las generadas automáticamente. También es posible llamar a la función demo_gui con argumentos de entrada (notar el uso de varargin).

Argumentos de entrada hObject, evendata y handles: Sirven para que todas las funciones internas se pasan información entre ellas. En concreto, handles es una estructura cuyos campos son los tags de los objetos gráficos y el valor de dichos campos es el número de handle asociado, por ejemplo,

```
handles =
       figure1: 173.0063
    popupmenu1: 187.0063
      uipanel1: 184.0063
```

```
           axes2: 179.0063
           axes1: 174.0063
       bot_salir: 186.0063
        bot_grid: 185.0063
```

La variable hObject contiene el número de handle de cada objeto. El usuario puede añadir más campos a la variable handle conteniendo, por ejemplo, datos de la aplicación. O también puede crear otra variable de tipo estructura independiente para contener los datos de la aplicación y dejar la variable handles sólo para los identificadores de los objetos.

8.2 Edición de la función *.m de la aplicación

Cabecera: La cabecera de la función de aplicación contiene unos comentarios de ayuda y unos comandos de inicialización que no es necesario editar:

```
function varargout = demo_gui(varargin)
% DEMO_GUI M-file for demo_gui.fig
%
%bla,bla,bla
%

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @demo_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @demo_gui_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
```

A continuación de la cabecera aparecen una serie de funciones:

Función OpeningFcn: Es la función que se ejecuta cuando se inicia la GUI. Notar que acepta parámetros de entrada adicionales por medio del argumento varargin.

En esta función se pueden generar y/o cargar los datos iniciales. También se puede llevar a cabo una primera representación gráfica.

En este primera function hay dos instrucciones que aparecen por defecto

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

La primera instrucción, `handles.output = hObject;`, añade el campo "output" a la variable handles y le asigna el handle correspondiente a la figure, hObject. La variable handles queda pues, así:

```
handles =
       figure1: 173.0013
         axes3: 6.0020
    popupmenu1: 182.0013
      uipanel1: 179.0013
         axes1: 174.0013
     bot_salir: 181.0013
      bot_grid: 180.0013
        output: 173.0013
```

Notar que handles.output (es decir, la ventana principal de la aplicación) será la salida mínima (por defecto) del fichero m (ver función OutputFcn más adelante).

La segunda instrucción, `guidata(hObject, handles);`, sirve para actualizar la estructura de *handles* si se han hecho cambios (por ejemplo, si hemos generado nuevos datos y los hemos guardado en nuevos campos de `handles`).

En nuestro ejemplo, usaremos la función OpeningFcn para generar un gráfico de inicio y representarlo. La función OpeningFcn quedará así

```
function demo_gui_OpeningFcn(hObject, eventdata, handles, varargin)
inicio(handles);
handles.output = hObject;
guidata(hObject, handles);
```

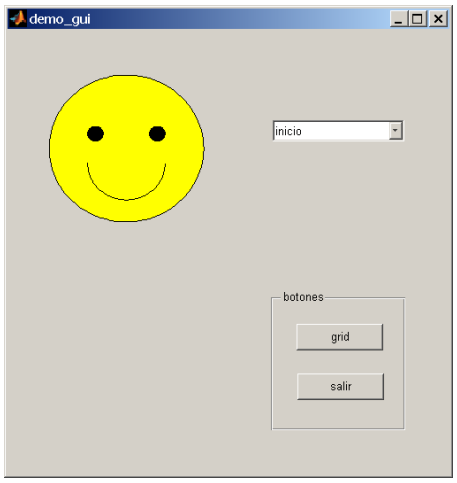
La subrutina inicio la definiremos al final del fichero m así:

```
function inicio(handles)
newplot;
x=linspace(0,2*pi);y=exp(j*x);
fill(real(y),imag(y),'y','Parent',handles.axes1);
hold(handles.axes1,'on')
o1=y*0.1-0.4+j*0.2;o2=y*0.1+0.4+j*0.2;
patch(real(o1),imag(o1),'k','Parent',handles.axes1);
patch(real(o2),imag(o2),'k','Parent',handles.axes1);
x=linspace(-pi,0);y=exp(j*x)*0.5-0.2*j;
plot(real(y),imag(y),'k','Parent',handles.axes1)
axis(handles.axes1,'off'),axis(handles.axes2,'off'),
hold(handles.axes1,'off')
```

Notar que hemos usado la propiedad 'Parent' para indicar en cuáles de los dos axes hay que realizar la representación. Notar también que hemos usado la función hold y la función axis indicando en cada caso a qué axes queríamos aplicar los cambios.

La instrucción newplot se ha añadido por precaución, para limpiar la figure, y evitar posibles superposiciones de gráficos.

Con este código, la ejecución inicial de la GUI da lugar a la siguiente pantalla:



Función OutputFcn: Es la función que se ejecuta como salida de la GUI. Notar que es posible que la GUI genere variables de salida adicionales (además del handle de la figure principal) adicionales por medio del argumento varargout. Esta función no la tocaremos en nuestro ejemplo.

```
function varargout = demo_gui_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;
```

Funciones de Callback de los controles:

Función asociada al botón bot_grid: En esta función ponemos los comandos que queremos que se ejecuten al clicar sobre el botón bot_grid (también podríamos haberlas escrito en el Property Inspector). Puesto que el callback es el comando grid a secas ello indica que cuando cliquemos en bot_grid, se pondrá y quitará la rejilla de manera alternativa.

```
function bot_grid_Callback(hObject, eventdata, handles)
grid(handles.axes1)
grid(handles.axes2)
```

Función asociada al botón bot_salir: En esta función podemos poner los comandos que queremos que se ejecuten al clicar sobre el botón bot_salir. En este ejemplo no pondremos nada puesto que ya hemos rellenado el campo Callback del property inspector de bot_salir con el comando “close”.

```
function bot_salir_Callback(hObject, eventdata, handles)
```

Funciones asociadas al pop up menu: Al crear el objeto pop up menú nos han aparecido dos funciones de callbacks: callback y createFcn. Sólo editaremos la primera. En ella, en función de la selección del pop up menú ejecutaremos una representación gráfica u otra, por ejemplo,

```
function popupmenu1_Callback(hObject, eventdata, handles)
val=get(hObject,'Value');
```

```
str=get(hObject,'String');
switch str{val}
    case 'bolas'
        [B1,B2]=bolas;
        rep_bolas(B1,B2,handles);
    case 'sombrero'
        S=sombrero;
        rep_sombrero(S,handles);
    case 'inicio'
        inicio(handles);
end

function popupmenu1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Al final del fichero hemos añadido las subrutinas que generar y representan las bolas y el sombrero. Éstas son:

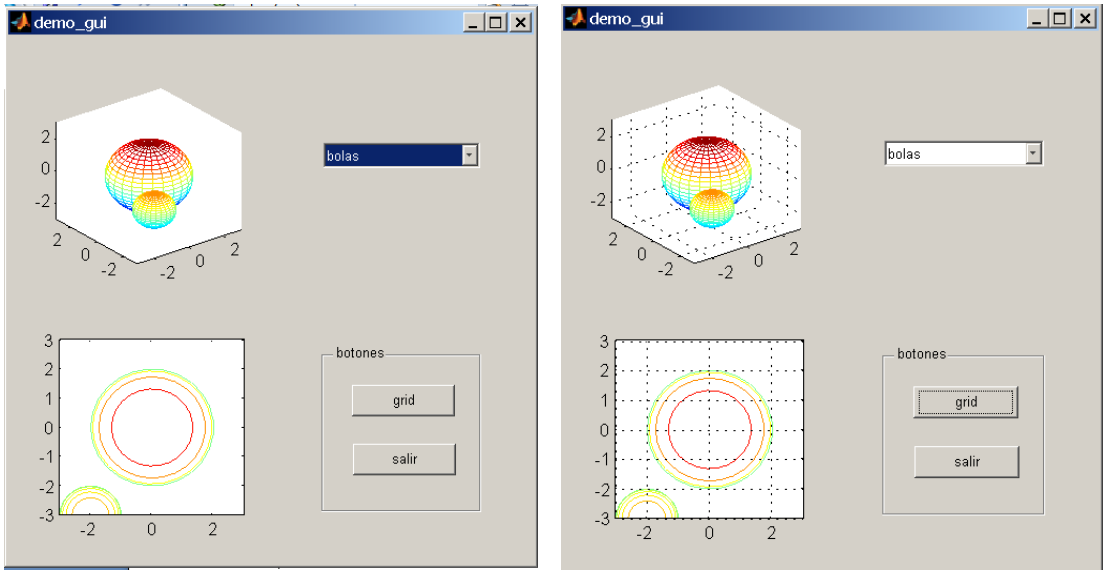
```
function [B1,B2]=bolas
[x,y,z]=sphere(25);
x1=x-2;y1=y-3;z1=z;
x2=x*2;y2=y*2;z2=z*2;
B1={x1,y1,z1};B2={x2,y2,z2};

function S=sombrero
warning('off','MATLAB:divideByZero');
[x,y]=meshgrid(-10:0.5:10,-10:0.5:10);
z=sin(sqrt(x.^2+y.^2))./sqrt(x.^2+y.^2);
S={x,y,z};

function rep_bolas(B1,B2,handles)
mesh(B1{1},B1{2},B1{3},'Parent',handles.axes1),
hold(handles.axes1,'on')
mesh(B2{1},B2{2},B2{3},'Parent',handles.axes1),
set(handles.axes1,'XLim',[-3 3],'YLim',[-3 3],'ZLim',[-3 3]);
hold(handles.axes1,'off');
%
contour(B1{1},B1{2},B1{3},'Parent',handles.axes2),
hold(handles.axes2,'on')
contour(B2{1},B2{2},B2{3},'Parent',handles.axes2),
set(handles.axes2,'XLim',[-3 3],'YLim',[-3 3]);
hold(handles.axes2,'off');
%
grid(handles.axes1,'off');grid(handles.axes2,'off');

function rep_sombrero(S,handles)
mesh(S{1},S{2},S{3},'Parent',handles.axes1);
contour(S{1},S{2},S{3},'Parent',handles.axes2),
grid(handles.axes1,'off');grid(handles.axes2,'off');
```

8.3 Ejecución de la aplicación

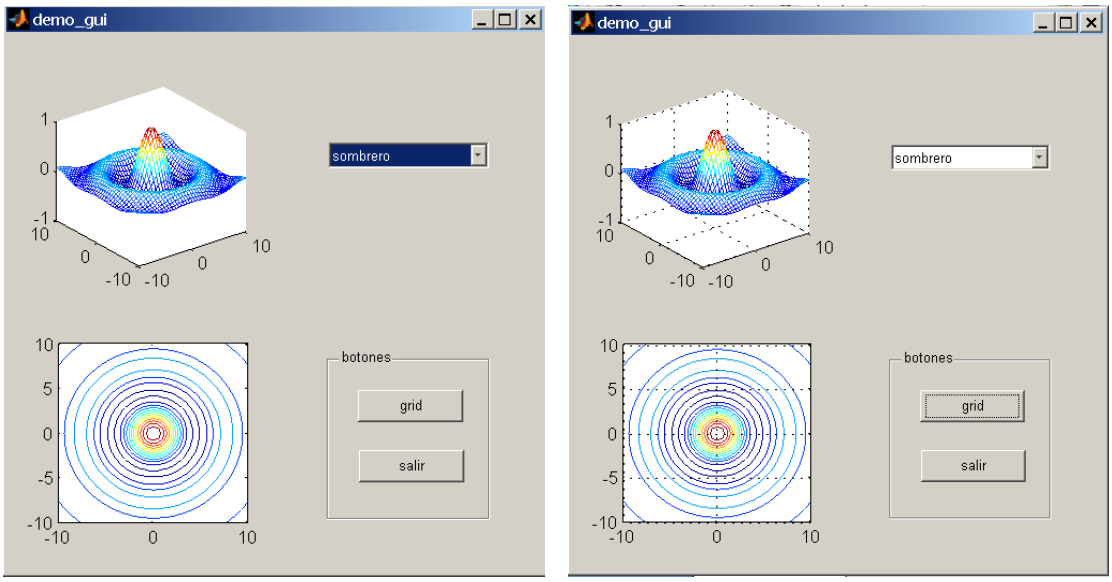
Para ejecutar la GUI basta con teclear

```
>> demo_gui
```

El resultado se muestra a continuación. Al seleccionar la opción “bolas” y clicar en el botón bot_grid el resultado es el siguiente:



Y al seleccionar la opción “sombrero”:



Y al clicar en el bot_salir se cierra la GUI.