

GUIs: Noções Gerais

Neste texto explica-se como implementar GUIs (graphic user interfaces). O objectivo será o de explicar como os GUIs podem ser implementados com conhecimentos mínimos de programação. Uma ferramenta de programação tornou a programação de GUIs muito simplificada, não necessitando de conhecimentos profundos como por vezes certos textos parecem fazer crêr. No entanto, convém antes de mais recordar alguns aspectos de programação.

Criação de Estruturas

Uma forma de agrupar informação com vários tipos de dados (por exemplo, matrizes, palavras ('strings') e inteiros) pode ser concretizada através da definição de estruturas. Uma estrutura é definida através de um nome e dos vários campos que a compõem. Por exemplo, podemos associar à estrutura denominada `disciplina1` vários campos: `disciplina1.semestre`, `disciplina1.ano`, `disciplina1.alunos`. Os dois primeiros campos guardam inteiros, enquanto que o último pode guardar um array de palavras ('strings').

Exemplo:

```
>> disciplina1.semestre=1
disciplina1 =
  semestre: 1
>> disciplina1.ano=3
disciplina1 =
  semestre: 1
  ano: 3
>> disciplina1.alunos={'Ricardo', 'Ana'}
disciplina1 =
  semestre: 1
  ano: 3
  alunos: {'Ricardo' 'Ana'}
```

No último caso, introduziu-se, por curiosidade o nome dos alunos. Foi definida uma matriz celular, importante para guardar elementos que sejam eles próprios vectores ou matrizes (o que é o caso: uma palavra é um vector de caracteres).

Natureza Local da Informação numa Função

Todas as variáveis que se definem dentro de uma função são, por defeito, locais. Isto quer dizer que assim que a execução da função terminar, os seus valores se perdem. Em geral, a passagem de informação entre diferentes funções do programa é feita através das variáveis de entrada e saída, explicitamente indicadas na linha da declaração da função. Este procedimento é uma segurança pois evita conflitos que poderiam surgir se variáveis com finalidades diferentes, usadas em diferentes funções do programa, tivessem o mesmo nome. Por isso, se se procurar desenvolver software que possa ser utilizado por outro programador, este procedimento é o geralmente adoptado. É por isso que mesmo na construção de GUIs, a informação sobre todos os objectos presentes no GUI está arquivada numa estrutura denominada **handles** que é passada para dentro de todas as funções, na linha de declaração.

Objectos Gráficos

Um GUI é composto por vários objectos. Alguns destes objectos, por exemplo os gráficos, são também eles próprios compostos de objectos. Por exemplo, uma figura é composta por um sistema de eixos e um menu. Por sua vez, no sistema de eixos estão incluídos outros objectos tais como imagens, linhas, texto, o sistema de cor e iluminação. A cada um destes objectos é associado um número (designado *handle*, i.e., pega) e uma estrutura de informação, contendo dados relativos a tudo o que pode ser necessário para o definir: a cor do background da zona exterior da figura, o tipo de letra no caso de um sistema de eixos, etc. A lista de *propriedades* que definem um objecto pode ser muito extensa. A capacidade para utilizar muitas das possibilidades depende da habilidade do programador. Importa conhecer três tipos de instruções:

1) Instruções para saber como aceder à estrutura de informação

Como conhecer o número identificador (handle) associado a um objecto? Quando se produz uma figura através de plot (surf, etc.) a função plot retorna o número de identificação. Por isso, se escrever `>> f=plot(x,y)`, `f` passará a ter esse número. Alternativamente pode-se utilizar a instrução `>> f=gcf` (i.e., get current figure) que fornece o handle da última figura alterada. Para aceder aos eixos pode utilizar `>> g= gca` (get current axes).

2) instruções para obter o valor de uma dada propriedade

Há muitas propriedades possíveis. Para listá-las pode usar duas estratégias: ou usar uma ferramenta que apresentaremos de seguida (o Property Inspector do GUIDE), ou então escrever `>> get(número identificador)` (em inglês, `get(handle)`). As propriedades são listadas, assim como os seus valores se existirem.

Exemplo:

```
>> f=plot([1 2],[1 3]);
>> get(f)
    Color: [0 0 1]
    EraseMode: 'normal'
    LineStyle: '-'
    LineWidth: 0.5000
    Marker: 'none'
    MarkerSize: 6
    MarkerEdgeColor: 'auto'
    MarkerFaceColor: 'none'
        XData: [1 2]
        YData: [1 3]
        ZData: [1x0 double]
    BeingDeleted: 'off'
    ButtonDownFcn: []
        Children: [0x1 double]
    Clipping: 'on'
    CreateFcn: []
    DeleteFcn: []
    BusyAction: 'queue'
    HandleVisibility: 'on'
    HitTest: 'on'
    Interruptible: 'on'
    Selected: 'off'
    SelectionHighlight: 'on'
    Tag: ''
    Type: 'line'
    UIContextMenu: []
    UserData: []
    Visible: 'on'
```

```
Parent: 151.0024
DisplayName: "
XDataMode: 'manual'
XDataSource: "
YDataSource: "
ZDataSource: "
```

Poderia querer conhecer só uma propriedade em particular. Para tal deve utilizar a instrução `get` mais completa: **`get(número identificador,'propriedade')`**.

Exemplo (continuação do caso precedente):

```
>> get(f,'MarkerSize')
ans =
     6
```

Pode passar o valor desta propriedade para uma variável. Para tal escreveria

```
>> Tamanho= get(f,'MarkerSize'); e a variável Tamanho passaria a ter o valor 6. No workspace poderia confirmar que a nova variável foi definida, bem como o seu tipo e valor.
```

Em vez de utilizar a função `get`, podemos reconhecer que a função `plot` tem associada uma estrutura de dados que pode ser acedida através do handle `f`. Assim, podemos alternativamente escrever `f.Color`, para aceder aos dados da cor, por exemplo.

3) instruções para modificar uma dada propriedade

Para modificar a propriedade desejada deve utilizar a instrução `set`:

`set(número identificador,'propriedade',novo valor)`

De notar que o novo valor deve ser um dado do tipo esperado para a propriedade. Por exemplo, em `MarkerSize`, o valor deverá ser um inteiro, enquanto que para uma cor se espera um vector 1x3 (com os valores entre 0 e 1 das componentes RGB da cor).

Exemplo (continuação do caso precedente):

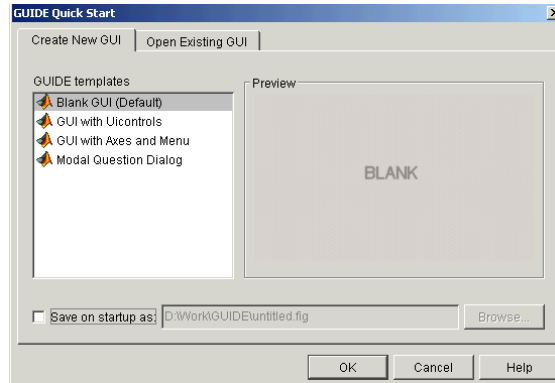
```
>> set(f,'MarkerSize',20)
>> set(f,'Color',[1 1 0])
```

Podemos alternativamente usar o facto de `f` ter uma estrutura de dados e escrever:

```
>> f.Color=[1 1 0]
```

A ferramenta GUIDE para a implementação de GUIs

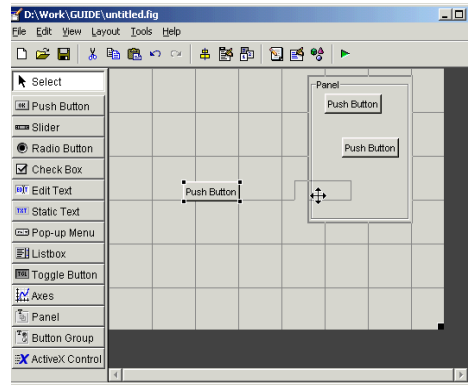
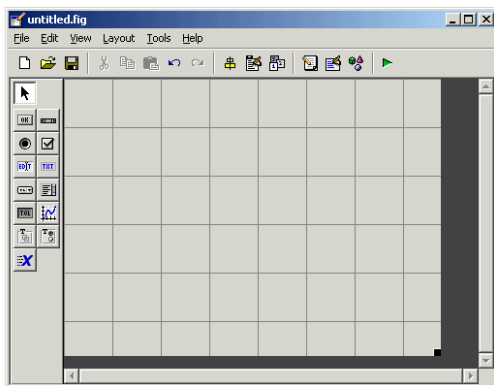
1) Inicializar GUIDE através de `>>guide`.



e escolher default, para não começar com GUIs já pré-definidos. Para abrir Guis anteriormente elaborados, usar ‘Open Existing GUI’.

2) Pode colocar o conjunto de botões que deseja na sua interface.

Se desejar que funcionem em bloco (por exemplo, há botões que podem ser exclusivos: se clicar num deles, elimina a escolha de outro) pode agrupá-los (usando ‘Panel’).




3) Aplicando um duplo clique sobre um objecto tem acesso ao ‘**Property Inspector**’ onde todas as propriedades do objecto estão listadas. Conhecê-las bem pode permitir atingir soluções muito versáteis (por exemplo: poderia aprender a fazer desaparecer um botão nunca determinada situação e a fazer aparecer outro, noutra, etc. No fim de ler estas notas tente fazê-lo!). No entanto, no âmbito desta disciplina estes aspectos mais elaborados serão acessórios deixando-se a cada um a vontade de os aprender. Por agora devemos focar algumas *Properties* realmente importantes:

- **Tag:** é o nome pelo qual o botão será reconhecido (o seu nome identificador).
- **Value:** valor associado ao estado do botão.
- **String:** palavra associada ao botão e que pode estar escrita no botão no écran.


Há botões para as quais certas propriedades não têm função, ainda que existam por defeito. Deverá trabalhar com as propriedades que serão logicamente úteis para cada objecto. Por exemplo, no caso de um *Slider Button* deve trabalhar com ‘*Value*’ – que corresponde à posição do botão, enquanto que num ‘*Edit Text*’ deve trabalhar com a propriedade ‘*String*’. No entanto deve notar que a propriedade ‘*Value*’ também se encontra definida para botões do tipo ‘*Edit Text*’. Assim se a modificar isso não trará nenhuma alteração visível no GUI. A existência destes campos redundantes pode no entanto ser útil, como veremos adiante, para guardar valores que podem ser lidos por funções independentes do programa (por exemplo, Callbacks; mais sobre isto mais à frente...).

Nalguns botões, como as *Check Box*, o ‘*Value*’ só pode adoptar dois valores - *Max* ou *Min* - consoante o estado. Isso não sucede no botão *Slider*, que pode variar numa gama de valores entre ‘*Min*’ e ‘*Max*’. ‘*Min*’ e ‘*Max*’ são propriedades destes objectos que podem ser alteradas directamente no Property Inspector, o que estabelecerá os seus valores por defeito quando o seu GUI fôr apresentado pela primeira vez ao utilizador. Podem também ser alteradas num programa com a instrução *set*, como se referiu atrás. O mesmo se aplica a outras propriedades, como o ‘*SliderStep*’ que estabelece o incremento no botão quando se clica sobre o botão ou quando se desloca o botão. Experimente testar estes conhecimentos!

Para mais detalhes, vá ao help do matlab e faça um search com uicontrol, onde poderá encontrar uma descrição de todas as propriedades possíveis associadas aos objectos.

4) Carregue em  para correr o GUIDE. Dê um nome ao ficheiro. Vai gerar um ficheiro .fig (que corresponde ao layout do GUIDE) e um .m (é onde vai estar o código que vai programar; note que tem de enviar ambos os ficheiros ao docente para que ele consiga correr o seu GUI, quando entregar trabalhos!). Abre-se também uma interface com os botões. Porém para eles se tornarem funcionais é necessário programar o ficheiro .m.

5) O ficheiro .m gerado parece extenso, embora isso se deva a todo um conjunto de comentários que podem ser de grande ajuda. Há funções com diversas finalidades, tais como iniciar a interface como um todo, ou iniciar os botões. Estes comandos não são essenciais até porque podem ser alterados com o Property Inspector no GUIDE, ou através de instruções *set*.

Devemo-nos concentrar na programação dos Callback’s associados aos botões. Os Callback são funções que serão executadas cada vez que um botão altera o seu valor por acção do utilizador (por exemplo, quando um botão é pressionado ou texto é inserido numa caixa de texto). Para se deslocar rapidamente para o Callback desejado no programa .m, pode pressionar  e seleccionar o Callback desejado.

Por exemplo, tendo-se alterado (no Property Inspector) o Tag associado ao Slider Button para Força (força), o nome do Callback correspondente foi alterado de forma correspondente:

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

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range
of slider
```

Há handles (pegas ou identificadores) para as várias propriedades do objecto com o nome Forca, que podem ser visualizadas durante a execução do programa, se utilizar o modo debug. Nesse caso pode colocar um ‘breakpoint’ numa determinada linha, e ao colocar o cursor sobre uma instrução ‘handles’, verá todo o conjunto de campos associado à estrutura ‘handles’ (pode também vê-lo através das variáveis apresentadas no workspace). Utilizando a instrução get podemos ler o valor de uma propriedade desejada:

```
x = get(hObject, 'Value')
```

Aqui hObject é o handle para o objecto na função de Callback que se está a analisar. Em cada função hObject é diferente, pois é uma variável de entrada, como pode verificar na linha de declaração. Podemos também alterar uma propriedade do botão, utilizando a instrução set, que tem a sintaxe set(H,'PropertyName',PropertyValue,...). Assim neste caso seria:

```
set(hObject, 'Value', 0.9)
```

que avançaria o cursor para outra posição.

A vantagem de utilizar a designação hObject, prende-se na concisão. Porém, e nesta disciplina será o procedimento mais adoptado, pode ler ou alterar uma propriedade de um objecto qualquer, usando a forma geral handles.nome_do_tag. Aqui, nome_do_tag é o tag do objecto presente no seu GUI.

6) Todas as propriedades dos objectos (botões, gráficos, etc.) existentes no painel podem ser alteradas e lidas em qualquer parte do programa. No entanto, isso requer ler a estrutura de handles que é um argumento passado para qualquer função de Callback. Nesse sentido comporta-se como uma estrutura a que todas as funções terão acesso.

Podemos alterar um campo na estrutura ‘handles’, correspondendo a propriedades de outros objectos presentes no GUI. Por exemplo, no Callback de um Slider Button, poderíamos ter, no ficheiro .m:

```
% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject      handle to slider1 (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of
slider

val= get(hObject,'Value');
set(handles.caixa,'String',num2str(val));
```

No final desta função, introduzimos duas instruções. A primeira lê para val a posição do botão (note-se como o comentários possuem Hints que nos explicam como proceder!). Depois, tendo mudado a Property Tag de uma caixa de Edit Text para caixa, podemos escrever nesta caixa o valor da posição do botão.

Como já se referiu a utilização do handle `hObject` é algo desnecessário, pois podemos sempre utilizar um procedimento usando a estrutura handles. No caso precedente, poderíamos ter escrito, em vez da primeira instrução:

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

O uso recorrente do tag associado aos diversos botões para alterar as suas propriedades torna conveniente que os seus tags tenham nomes facilmente memorizáveis. Por isso, uma das tarefas importantes no início da construção do Gui será a de mudar o nome das tags, alterar os valores e strings com que os botões são inicializados por defeito (lembre-se: um Gui é um Graphical User Interface e deve ser desenvolvido para que qualquer utilizador possa utilizar o seu código com conhecimentos mínimos; o Gui deve por isso ser tão auto-explicativo quanto possível!).

Na estrutura handles podemos ainda acrescentar campos para guardar variáveis que possam ser necessárias noutras funções do programa. Por exemplo na função de Callback do botão slider da Força podíamos definir o handle com campo ForcaX, se o valor desse botão significasse a componente x da força, usando:

```
handles.ForcaX=get(hObject,'Value');
```

Para que este novo campo possa ser guardado na estrutura handles onde está toda a informação do seu Gui, deve utilizar a instrução guidata:

```
guidata(hObject,handles);
```

que actualiza a estrutura dos handles (note que de outra forma esta informação perder-se-ia quando terminasse a execução do callback). Assim, após utilizar o guidata neste momento, se noutra função do programa, escrevesse:

```
fx=handles.ForcaX;
```

poderia aceder ao valor da componente da força segundo x, colocando-a na nova variável fx (que, em principio, será uma variável local à função onde está a ser definida).

7) Cada gráfico tem um handle. Se tiver mais que um gráfico no seu GUI, terá que especificar em que gráfico quer que seja feito um plot (por exemplo), usando o handle correspondente. Por exemplo, `plot(handles.grafico1, x,y);` faz o `plot(x,y)` no grafico com tag `grafico1`.

As **ideias importantes** são:

- Cada objecto (janela, gráfico, botão) tem um handle (pega/identificador) em que o campo tem o seu nome. Por exemplo, uma janela chamada `simulacao1`, tem um handle designado `handle.simulacao1`.

- No caso de objectos, os handles têm propriedades associadas que podem ser alteradas com a instrução

```
set(handle, 'propriedade', novo valor da propriedade)
```

e lidas com

```
get(handle, 'propriedade')
```

No gui em geral o handle é dado por: `handles.nome_do_tag`

- Há variáveis que podem ser guardadas na estrutura de handles, e podem ser acedidas em qualquer parte do programa, como se de uma variável global se tratassem.

- a estrutura dos handles tem de ser actualizada a nível global, através da instrução `guidata`, dado que em cada Callback handles entra como variável local na definição da função.

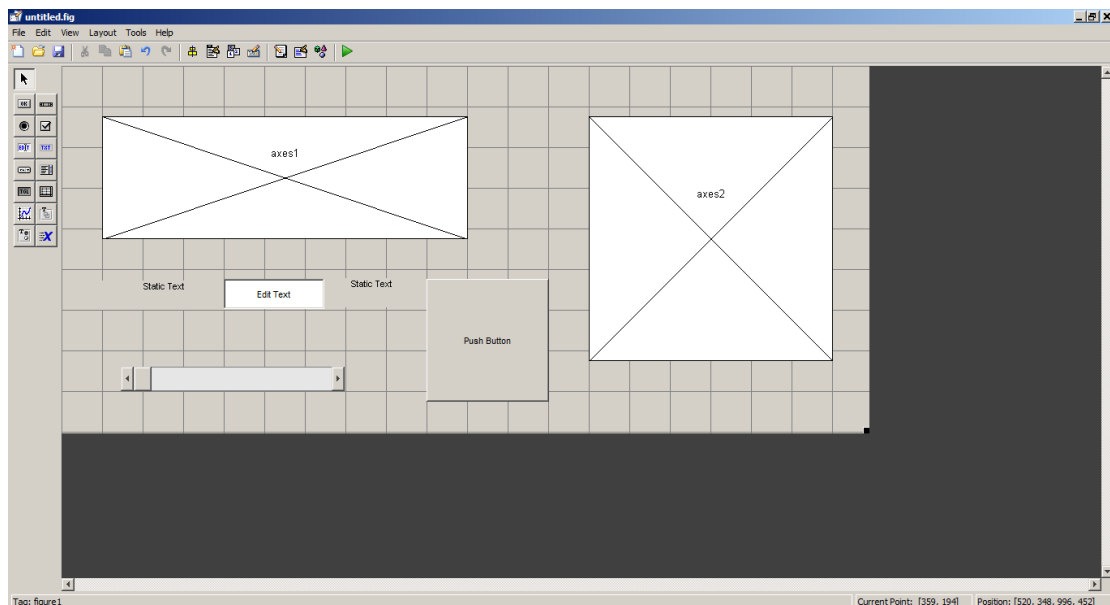
Exercício de Aplicação: STEP BY STEP

PROBLEMA: Imagine-se que queremos simular o movimento de um oscilador harmónico: uma massa presa a uma mola, presa a uma parede. Além disso queremos representar o gráfico de $x(t) = A \cos\left(\frac{2\pi t}{T} + \varphi\right)$, à medida que for realizada a animação.

STEP 1: Iniciar Guide

STEP 2: Definir no GUIDE todos os objectos necessários.

Precisamos de 2 gráficos (uma para a animação e outro para o gráfico de $x(t)$), e ainda de botões que possam alterar a amplitude, o período e a fase inicial. Podemos começar por criar a seguinte disposição de objetos:



Temos dois gráficos, duas caixas de texto estático para legendar a caixa de texto de edição que o utilizador poderá alterar, um slider button que utilizaremos para alterar facilmente a fase inicial, e um botão de pressão para iniciar a simulação.

Ainda não introduzimos as restantes caixas de texto, pois estas poderão ser obtidas das anteriores fazendo copy&paste, o que permitirá que fiquem todas com iguais dimensões e formatações. Nalgumas versões do Matlab há ferramentas para alinhar os objectos que também poderão ajudar a complementar esse tipo de tarefas.

STEP 3: Aplicando um duplo click sobre cada objecto, aparece uma caixa designada Property Inspector onde estão listadas todas as propriedades associadas a cada objecto. Como já se referiu, algumas destas propriedades aparecem por defeito para cada objecto, embora não sejam por eles utilizados. Por exemplo, pode confirmar que numa caixa de texto há um campo designado 'Value' que não será utilizado por este objecto, enquanto que num Slider Button, há um campo também redundante designado 'String'. No entanto, o campo 'String' é essencial para o funcionamento de uma caixa de texto (correspondendo ao texto que aparece no monitor) e o campo 'Value' é utilizado para definir a posição de um Slider Button.

Neste passo deve percorrer todos os objectos e, através do Property Inspector para cada objecto, modificar os campos:

- **Tag:** deve escolher nomes elucidativos que ajudem a identificação do objecto. Neste exemplo alteraram-se os Tags dos graficos para animacao e grafico; o edit text box ficou com a tag amplitude; o slider button, fase; o push button ficou com a tag acao (não use caracteres especiais, como acentos e cedilhas). Optou-se ainda por não alterar as tags das caixas de texto estático, dado que não serão utilizadas a nível de programação.

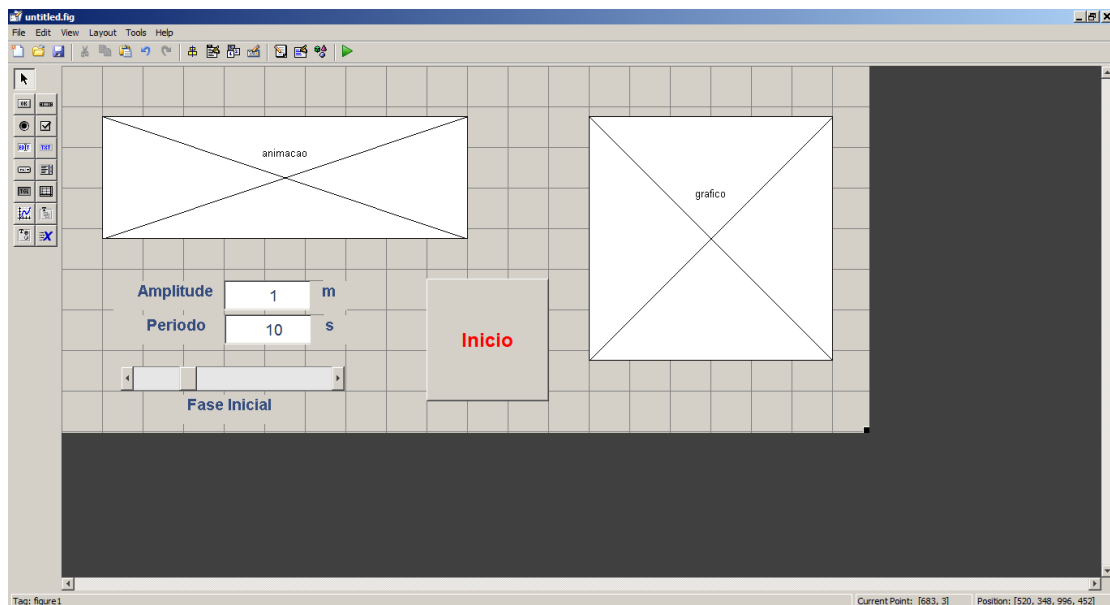
- **Value no Slider Button:** alterou-se o valor inicial, para 90.


- **Max no Slider Button:** alterou-se para o valor 360, porque a fase inicial está entre 0 e 360°.

- **String nas Static and Edit Boxes:** nas caixas 'Static Text' que servem para legendar introduziu-se o texto 'Amplitude =', 'm'. Nas caixas 'Edit Text' introduziu-se um valor adequado por defeito: '1'.

- **Push Button:** no Push Button, a String passou a ser 'Inicio'.

Nos property inspectors dos vários botões, alteraram-se ainda as cores e dimensões das letras. As caixas de texto em falta para o periodo foram tambem introduzidas (usando copy & paste) e inicializadas como as outras. No fim destes passos o GUI apresenta o seguinte aspecto:



STEP 4: Correr o GUIDE pressionando em  e analisar o resultado gráfico. Será necessário gravar o ficheiro com um nome. Optou-se por 'osciladorHarmonico'. São então criados dois ficheiros: osciladorHarmonico.m e osciladorHarmonico.fig. Pode ainda voltar ao GUIDE e corrigir algum detalhe de apresentação (posicionamento de botões, dimensões, etc.) e voltar ao **STEP 4**.

STEP 5: Ir ao ficheiro .m e programar Callbacks. Inicialmente programamos somente o callback Inicio. No entanto, sempre que o utilizador alterar uma caixa de edição de texto, ao carregar no enter os callbacks destas caixas serão chamados, pelo que linhas de código poderiam também ser introduzidas aí.

No callback Inicio podemos incluir todo o código:

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

A= str2num(get(handles.amplitude,'string')); % lê strings das caixas de
                                           %texto e converte-as em numeros

T= str2num(get(handles.periodo,'string'));
fase= get(handles.fase,'value');          % lê o campo value, que já é um numero
faseRad= fase*2*pi/360;                    % conversão de graus para radianos
tmax=100;                                  % tempo máximo
dt= 0.2;                                   % vamos apresentar a simulacao de 0.2 em 0.2 segundos
i=1;
x0=A+1;                                    % para que a massa não bata na parede, incluimos um x0
                                           % que é uma unidade superior a A.

for t=0:dt:tmax
    x(i)= x0+A*cos(2*pi/T*t+faseRad); % i-ésimo valor de x é guardado no
                                           % i-ésimo elemento dos vetores x e t
    tempo(i)=t;
    i=i+1;
end

for i=1:length(tempo)                    % faz a simulação
    plot(handles.animacao,[0 0 max(x)*1.1],[1 0 0],'-','linewidth',4)
    xlim(handles.animacao,[-0.1 max(x)*1.1])
    ylim(handles.animacao,[-0.5 2])
    xx=linspace(0,x(i),100);
    lambda=(x(i)-0)/10;
    yy=0.15*sin(2*pi*xx/lambda)+0.25; % representação da mola como uma onda
                                           %com n° de maximos constante, ou seja c.d.o. variável.
    hold (handles.animacao,'on')
    plot(handles.animacao,xx,yy,'-r');
    plot(handles.animacao,x(i),0.3,'ro','Markersize',30,'MarkerFaceColor','r')
    hold(handles.animacao,'off')
    plot(handles.grafico,tempo(1:i),x(1:i),'.b')
    xlim(handles.grafico,[0 50]);
    ylim(handles.grafico,[min(x) max(x)]);
    pause(dt)
end
```

Nas primeiras instruções lê-se o valor das caixas e do slider button. Note que houve necessidade de converter um número para uma string, através da instrução `num2str`. Caso fosse necessário fazer a operação inversa – alterar o número apresentado por uma string numa edit box – ter-se-ia que utilizar o comando `set` e converter o número numa string, usando o comando inverso `str2num`.

No código apresentado houve uma ligeira preocupação em tornar os valores máximos dos eixos flexíveis, de forma a que se adaptem aos diferentes valores da amplitude que o utilizador possa escolher. No entanto, é sempre possível melhorar. Assim sugere-se que o aluno, a título de exercício:

- 1) legende convenientemente o gráfico, com as instruções `xlabel`, `ylabel`, `title`...
- 2) introduza um título no Gui,
- 3) introduza uma caixa de texto estática onde apareça o valor numérico da fase sempre que o utilizador altere a posição do slider button da fase inicial.

Nota: para tal terá que programar o callback do `sliderbutton`; neste deve ler o valor do `slider button` e escrever o seu valor na nova caixa de texto.

