

Des fonctions MATLAB à IHM

Par DIMITRI PIANETA

Version 1

(2014-2015)

Chapitre 1 : Fonctions de bases

Matlab nous donne quelques fonctions bases.

Fonction	Description
+	addition
-	Soustraction
*	Multiplication
^	Exposant ex : 1.3^3.2
Trigonometric Functions	
sin(x)	Retourne le sinus de x
sind(x)	Retourne le sinus de x degrés
cos(x)	Retourne le cosinus de x
cosd(x)	Retourne le cosinus de x degrés
tan(x)	Retourne la tangente de x
tand(x)	Retourne la tangente de x degrés
atan(x)	Retourne la tangente inverse de x
atand(x)	Retourne la tangente inverse de x en degrés
acos(x)	Retourne le cosinus inverse de x
acosd(x)	Retourne le cosinus inverse de x en degrés
asin(x)	Retourne le sinus inverse de x
asind(x)	Retourne le sinus inverse de x en degrés
Mathematical Functions	
exp(x)	Retourne e^x
log(x)	Retourne le logarithme naturel de x
log10(x)	Retourne le $\log_{10}(x)$
sqrt(x)	Retourne la racine carrée de x
abs(x)	Retourne la valeur absolue de x
round(x)	Retourne le x dans l'intervalle fermé
ceil(x)	Retourne le plus petit entier plus grand ou égale à x
floor(x)	Retourne le plus grand entier moins grand ou égale à x
isprime(n)	Retourne true si n est prime
factor(k)	Retourne un factoriel de k
sign(x)	Retourne le signe de (1 ou -1) de x
rand	Retourne un nombre pseudo aléatoire entre 0 et 1
rand(m)	Retourne une matrice de m x m de nombres aléatoires
rand(m,n)	Retourne une matrice de m x n de nombres aléatoires

Chapitre 2 : String, vecteurs

1) String

```
firstname='Alfonso';  
lastname='Bedoya';  
idea1='Buy low, sell high';
```

Affichages:

```
>> disp(firstname)  
Alfonso  
>> disp(lastname)  
Bedoya
```

Concaténations de string

```
>> fname=[firstname,lastname];  
>> disp(fname)  
AlfonsoBedoya  
>> fullname=[firstname,' ',lastname];  
>> disp(fullname)  
Alfonso Bedoya  
>> disp([idea1, ', young ', firstname, '! ] );  
Buy low, sell high, young Alfonso!
```

Quelques methods utiles:

Fonctions	Descriptions
num2str(x)	Retourne un string correspondant au nombre stocker en x
str2num(s)	Retourne un nombre correspondant au string s
str2double(s)	Retourne un nombre correspondant à string s
length(s)	Retourne le nombre de caractère de string
lower(s)	Retourne le string s pour un exposant
upper(s)	Retourne le string s pour un indice
sName(4)	Retourne le 4ième caractère de string s
sName(4:6)	Retourne le 4ième au 6 ^{ème} caractère dans s

Pour récupérer une chaîne de caractère

```
nYears=input('Enter the number of years: ');
```

```
firstName=input('Please enter your first name: ','s');
```

2) Vecteur

```
>> vp=[1, 4, 5, 9];  
>> disp(vp)  
1 4 5 9
```

```
>> vp(1)=47;  
>> vp(3)=1.2;  
>> disp(vp)  
47.0000 4.0000 1.2000 9.0000
```

Fonctions spécifiques

ones(m,n) : une matrice de 1 de dimension m x n

zeros(m,n) : une matrice de 0 de dimension m x n

Chapitre 3 : Le plot

La commande de base :

```
plot(<vector of x-values>,<vector of y-values>,<style-option string>)
```

Exemple :

```
v1=[1, 3, 4, 6, 5, 2];
v2=[1, 2, 2, 3, 4, 2];
plot(v1,v2,'-o');
axis([0 7 0 5]);
```

Les paramètres options du plot

Spécification	Couleur Ligne	Spécification	Marker
r	Rouge	.	point
g	Vert	o	Cercle
b	Bleu	x	Croix
c	Cyan	+	Plus
m	Magenta	*	Asterix
y	Jaune	s	Carré
k	Noir	d	Losange
w	Blanc	v	Triangle en bas
Spécification	Style de ligne	^	Triangle en haut
-	Solide	<	Triangle à gauche
--	Dashed	>	Triangle à droite
:	Deux points	p	Pentagone
..	Deux points et le point	h	Hexagone

La fonction :

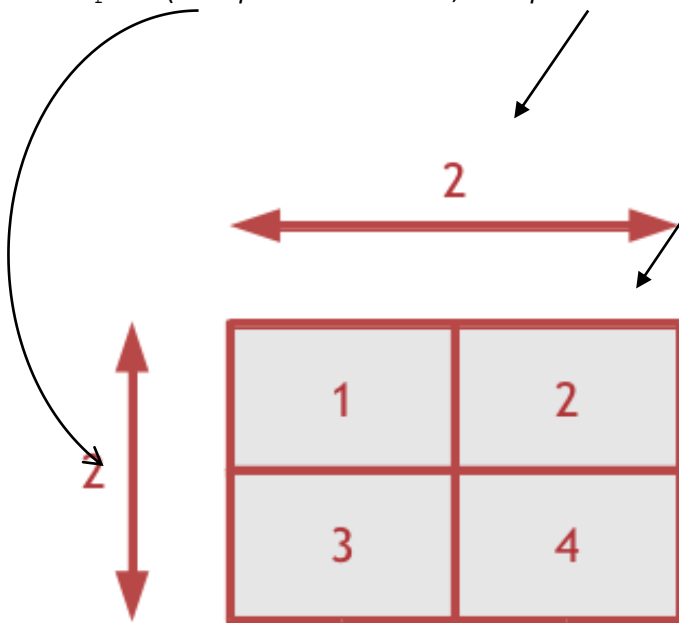
```
axis([xmin, xmax, ymin, ymax])
```

axis manual	freezes the scaling at the current limits
axis auto	returns the axis scaling to its default, automatic mode
axis tight	sets the axis limits to the range of the data
axis equal	sets the aspect ratio so that tick mark increments on the x, y, and z axes are equal in size
axis square	makes the current axis box square in size
axis normal	undoes the effects of axis square and axis equal
axis off	turns off all axis labeling, tick marks, and background
axis on	turns axis labeling, tick marks, and background back on
grid on	draws grid lines on the graph

Chapitre 4 : Le subplot

La syntaxe est la suivante:

`subplot (Nbre pavés sur hauteur, Nbre pavés sur largeur, Numéro pavé)`



Exemple de code :

```
x = 0: 2*pi/100 : 2*pi;
subplot(221)
plot(x, sin(x))
subplot(222)
plot(x, cos(x), x, sin(x), '-. ')
subplot(223)
plot(cos(x), sin(x))
subplot(224)
plot(sin(2*x), sin(3*x))
```

Chapitre 5 : Bar graphs and histogrammes

Baton vertical

```
%% makebar1.m
% demonstrate the use of a bar plot
%% set data
temperatures=[71, 80, 73, 72, 78, 81, 73, 76];
%% plot data as bar chart
bar(temperatures);
xlabel('Measurement');
ylabel('Temperature(F)');
grid on;
```

Histogramme

La fonction hist

Chapitre 6 : Les boucles, conditions

IF:

```
if <logical expression>
<block of statements>
end
```

Exemple:

```
% ifdemo.m
ageThreshold=50;
age=input('Please enter your age: ');
if age>ageThreshold
disp(' Wow, getting up there. ');
end
disp('Thanks!')
```

```
if <logical expression>
<block1>
else
<block2>
End
```

Exemple :

```
% PrimeCheck.m
% prompt for number, check if it's prime or not
% using isprime function
% Author: B.J. Honeycutt
%
disp(' ');
disp('---- Prime number checker ----');
n=input('Enter number to check: ');
if isprime(n)
disp(['Yes! ',num2str(n),' is prime!']);
else
disp(['No, ',num2str(n),' is not prime.']);
end
disp(' ');
```

```
if <logical expression 1>
<block1>

elseif <logical expression 2>

<block2>
elseif <logical expression 3>
<block3>
elseif <logical expression 4>
<block4>
else
<block5>
end
```

Exemple:

```
if (x >= -100)&&(x <= -5)
f(ix)=0;
elseif (x > -5)&&(x <= 0)
f(ix)=1;
elseif (x > 0)&&(x <= 5)
f(ix)= -1;
elseif (x > 5)&&(x <= 100)
f(ix)=0
else
disp('Error—function out of range [-100,100]');
end
```

Expressions logiques :

&& AND

|| OR

& element-by-element AND for arrays

| element-by-element OR for arrays

~ not

== is equal to

~= is not equal to

> is greater than

< is less than

>= is greater than or equal to

<= is less than or equal to

FOR LOOPS :

```
for <index>=<initial value>:<final value>
<block>
end
```

Exemple :

```
N=7;
for k=1:N
disp(['k= ',num2str(k)]);
end
disp('Done!');
```

WHILE LOOPS

```
while <condition>
<block>
end
```

Exemple :

```
% continue to add up whole numbers until total
% exceeds, or is equal to, maxSum
maxSum=200;
tot=0;
k=0;
while tot < maxSum
```

```
k=k+1;
tot=tot+k;
end
disp(['Sum of first ',num2str(k),' integers = ',num2str(tot)]);
```

SWITCH

```
switch monthStr
    case 'February'
        nDays=28;
    case {'September', 'April', 'June', 'November'}
        nDays=30;
    otherwise
        nDays=31;
end
```

Chapitre 7 : La gestion des paramètres de figure

Il est possible de mettre un texte sur une courbe.

Par exemple comme ce code:

```
>> figure(1)
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
>> text(pi,0,' \leftarrow sin(\pi)', 'FontSize',18, 'Color', [1 0 0], 'FontName', 'arial' )
```

Descriptions du code:

La syntaxe générale est la suivante :

```
text(x,y,'string')
```

```
text(x,y,'string',propriétés)
```

Ici on veut mettre $y = \sin(\pi)$, puis la taille de la police 18, ensuite la couleur ('Color'), on finit par la police (FontName).

Color:

Voici une liste non exhaustif des couleurs:

[1 1 0]	y	yellow	Jaune
[1 0 1]	m	magenta	Magenta
[0 1 1]	c	cyan	Cyan
[1 0 0]	r	red	Rouge
[0 1 0]	g	green	Vert
[0 0 1]	b	blue	Bleu
[1 1 1]	w	white	
[0 0 0]	k	black	

Il est aussi possible de changer le fond de la figure en utilisant la commande **whitebg**.

Cette fonction change toute les plots de la figure.

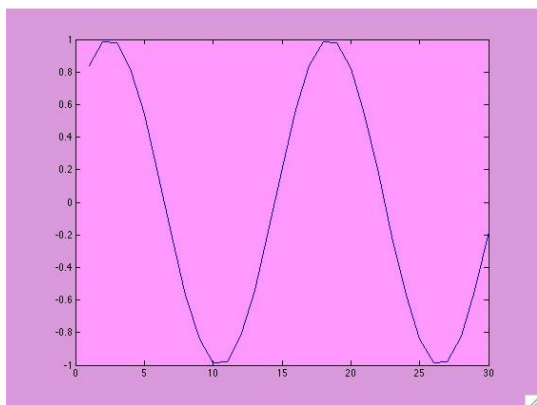
Un petit exemple :

```
>> t = -pi:0.1:pi;
```

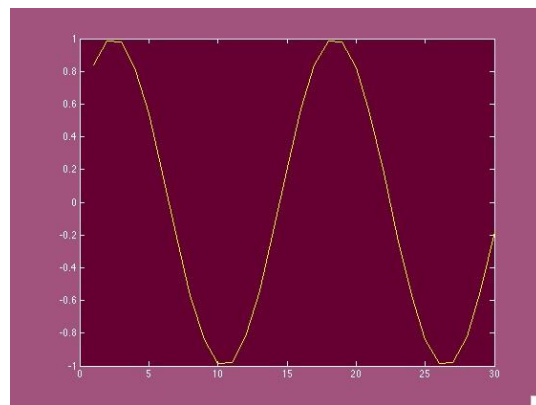
```
>> plot(t,y)
```

```
>> whitebg([1 1 1])
```

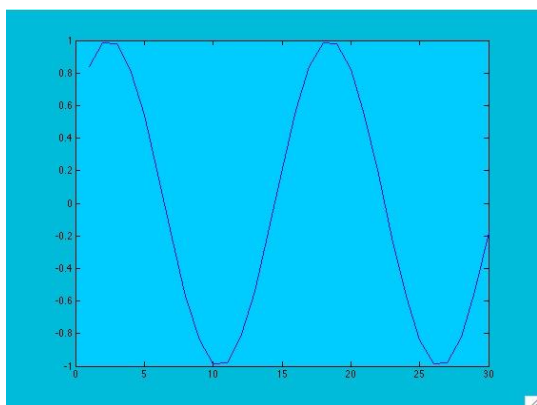
```
whitebg([255/255 153/255 255/255])
```



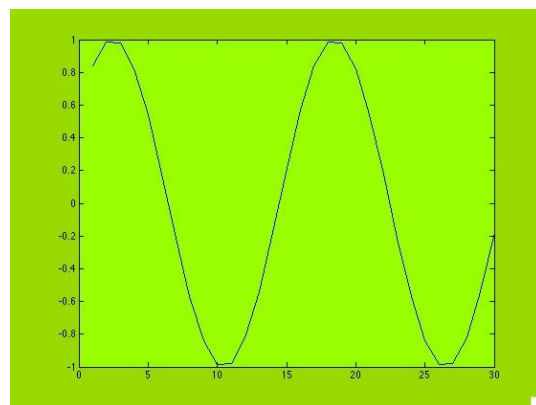
```
whitebg([102/255 0/255 51/255])
```



```
whitebg([0/255 204/255 55/255])
```



```
whitebg([153/255 255/255 0/255])
```



- Si on reprend exemple vu au début de ce paragraphe:

```
>> figure(1)
```



```
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
```

```
>>text(pi,0,' \leftarrow \sin(\pi)', 'FontSize',18, 'Color', [1 0 0], 'FontName', 'arial' , 'EdgeColor','red')% EdgeColor permet de créer un cadre ici rouge
```

On peut rajouter à la suite d'autres attributs comme:

- **BackgroundColor** : Color of the rectangle's interior (none by default).
- **LineStyle** : Style of the rectangle's edge line (first set EdgeColor).
- **LineWidth** : Width of the rectangle's edge line (first set EdgeColor)
- **Margin** : Increase the size of the rectangle by adding a margin to the existing text extent rectangle.

- Il est aussi possible de changer les traits de traçage dans un plot.

Voici la syntaxe:

```
plot(x, y, 'Color', [0.5, 1.0, 0.0], 'LineStyle', '--')
```

Des paramètres listés:

- **Extent**: position rectangle (read only)

Position and size of text. A four-element read-only vector that defines the size and position of the text string.

```
[left,bottom,width,height]
```

If the `Units` property is set to `data` (the default), `left` and `bottom` are the x and y coordinates of the lower left corner of the `text` `Extent`.

For all other values of `Units`, `left` and `bottom` are the distance from the lower left corner of the axes position rectangle to the lower left corner of the `text` `Extent`. `width` and `height` are the dimensions of the `Extent` rectangle. All measurements are in units specified by the `Units` property.

- **FontAngle**: {normal} | italic | oblique

Exemple :

```
>> plot(0:pi/20:2*pi,sin(0:pi/20:2*pi))
```

```
>> text(pi,0,' \leftarrow \sin(\pi)', 'FontSize',18, 'Color', [1 0 0], 'FontName', 'arial', 'FontAngle', 'italic', 'EdgeColor','red')
```

```
>> title('Graphe exemple', 'FontAngle', 'oblique')
```

- **FontName**: Pour savoir les polices de font disponible, utiliser la commande `listfonts`
- **FontSize** : C'est la taille de la police selon le `FonUnits` (par défaut 10 points)
- **FontWeight** : light | {normal} | demi | bold
- **FontUnits**: {points} | normalized | inches | centimeters | pixels
- **HorizontalAlignment** : {left} | center | right

HorizontalAlignment viewed with the VerticalAlignment set to middle (the default).

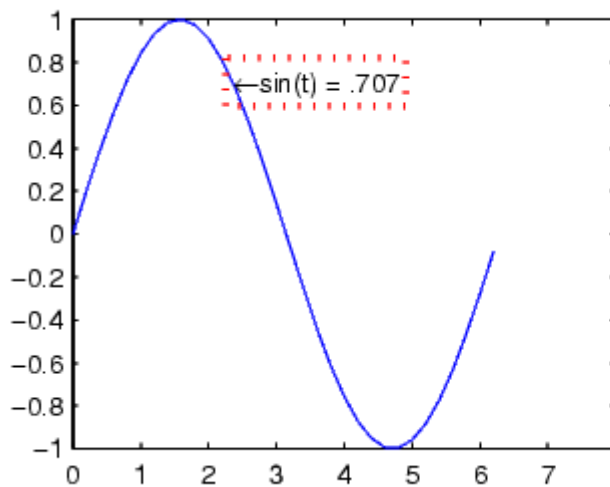


- **LineStyle** : {-} | -- | : | -. | none

Symbol	Line Style
-	Solid line (default)
--	Dash line
:	Dotted line
-.	Dah-dot line
none	No line

Par exemple comme code:

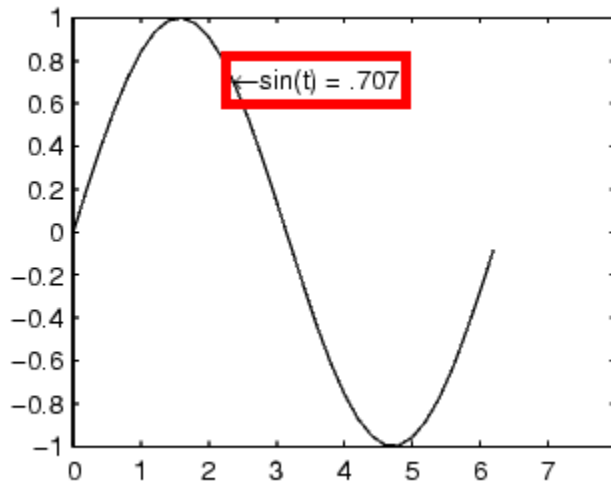
```
text(3*pi/4, sin(3*pi/4), ...
    '\leftarrow sin(t) = .707', ...
    'EdgeColor', 'red', ...
    'LineWidth', 2, ...
    'LineStyle', ':');
```



- **LineWidth** : scalar(points)

Code exemple:

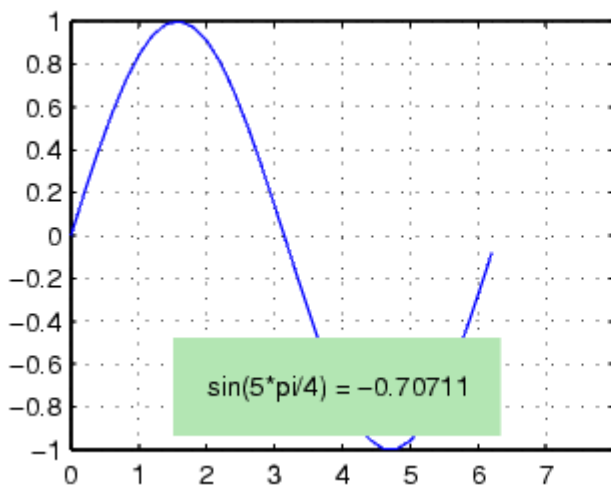
```
text(3*pi/4, sin(3*pi/4), ...
    '\leftarrow sin(t) = .707', ...
    'EdgeColor', 'red', ...
    'LineWidth', 3);
```



- **Margin** : scalar(pixels)

Code exemple:

```
text(5*pi/4, sin(5*pi/4), ...
    ['sin(5*pi/4) = ', num2str(sin(5*pi/4))], ...
    'HorizontalAlignment', 'center', ...
    'BackgroundColor', [.7 .9 .7], ...
    'Margin', 10);
```



- **Parent** : handle

Text object's parent. The handle of the text object's parent object. The parent of a text object is the axes in which it is displayed. You can move a text object to another axes by setting this property to the handle of the new parent.

- **Position** : [x,y,[z]]

Location of text. A two- or three-element vector, $[x \ y \ [z]]$, that specifies the location of the text in three dimensions. If you omit the z value, it defaults to 0. All measurements are in units specified by the `Units` property. Initial value is $[0 \ 0 \ 0]$.

- **Rotation** : scalar(défaut = 0)

Text orientation. This property determines the orientation of the text string. Specify values of rotation in degrees (positive angles cause counterclockwise rotation).

- Tableaux des caractères spéciaux:
-

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
<code>\alpha</code>	α	<code>\upsilon</code>	υ	<code>\sim</code>	\sim
<code>\beta</code>	β	<code>\phi</code>	ϕ	<code>\leq</code>	\leq
<code>\gamma</code>	γ	<code>\chi</code>	χ	<code>\infty</code>	∞
<code>\delta</code>	δ	<code>\psi</code>	ψ	<code>\clubsuit</code>	\clubsuit
<code>\epsilon</code>	ϵ	<code>\omega</code>	ω	<code>\diamondsuit</code>	\diamondsuit
<code>\zeta</code>	ζ	<code>\Gamma</code>	Γ	<code>\heartsuit</code>	\heartsuit
<code>\eta</code>	η	<code>\Delta</code>	Δ	<code>\spadesuit</code>	\spadesuit
<code>\theta</code>	θ	<code>\Theta</code>	Θ	<code>\leftrightharpoon</code>	\leftrightarrow
<code>\vartheta</code>	ϑ	<code>\Lambda</code>	Λ	<code>\leftarrow</code>	\leftarrow
<code>\iota</code>	ι	<code>\Xi</code>	Ξ	<code>\uparrow</code>	\uparrow
<code>\kappa</code>	κ	<code>\Pi</code>	Π	<code>\rightarrow</code>	\rightarrow
<code>\lambda</code>	λ	<code>\Sigma</code>	Σ	<code>\downarrow</code>	\downarrow
<code>\mu</code>	μ	<code>\Upsilon</code>	Υ	<code>\circ</code>	\circ
<code>\nu</code>	ν	<code>\Phi</code>	Φ	<code>\pm</code>	\pm
<code>\xi</code>	ξ	<code>\Psi</code>	Ψ	<code>\geq</code>	\geq
<code>\pi</code>	π	<code>\Omega</code>	Ω	<code>\propto</code>	\propto

<code>\rho</code>	ρ	<code>\forall</code>	\forall	<code>\partial</code>	∂
<code>\sigma</code>	σ	<code>\exists</code>	\exists	<code>\bullet</code>	\bullet
<code>\varsigma</code>	ς	<code>\ni</code>	\ni	<code>\div</code>	\div
<code>\tau</code>	τ	<code>\cong</code>	\cong	<code>\neq</code>	\neq
<code>\equiv</code>	\equiv	<code>\approx</code>	\approx	<code>\aleph</code>	\aleph
<code>\Im</code>	\Im	<code>\Re</code>	\Re	<code>\wp</code>	\wp
<code>\otimes</code>	\otimes	<code>\oplus</code>	\oplus	<code>\oslash</code>	\oslash
<code>\cap</code>	\cap	<code>\cup</code>	\cup	<code>\supseteq</code>	\supseteq
<code>\supset</code>	\supset	<code>\subseteq</code>	\subseteq	<code>\subset</code>	\subset
<code>\int</code>	\int	<code>\in</code>	\in	<code>\circ</code>	\circ
<code>\rfloor</code>	\rfloor	<code>\lceil</code>	\lceil	<code>\nabla</code>	∇
<code>\lfloor</code>	\lfloor	<code>\cdot</code>	\cdot	<code>\ldots</code>	\dots
<code>\perp</code>	\perp	<code>\neg</code>	\neg	<code>\prime</code>	\prime
<code>\wedge</code>	\wedge	<code>\times</code>	\times	<code>\emptyset</code>	\emptyset
<code>\rceil</code>	\rceil	<code>\surd</code>	\surd	<code>\mid</code>	\mid
<code>\vee</code>	\vee	<code>\varpi</code>	ϖ	<code>\copyright</code>	\copyright
<code>\langle</code>	\langle	<code>\rangle</code>	\rangle		

Exemple de syntaxe:

```
title("\bf \leftarrow Graphe exemple', 'FontAngle', 'oblique')
```

On peut toujours rajouter :

- `\bf` -- bold font
- `\it` -- italics font
- `\sl` -- oblique font (rarely available)
- `\rm` -- normal font
- `\fontname{fontname}` -- specify the name of the font family to use.
- `\fontsize{fontsize}` -- specify the font size in FontUnits.

- **Tag** : string

User-specified object label. The `Tag` property provides a means to identify graphics objects with a user-specified label. This is particularly useful when constructing interactive graphics programs that would otherwise need to define object handles as global variables or pass them as arguments between callback routines. You can define `Tag` as any string.

- **Units** : pixels | normalized | inches | centimeters | points | {data}

Units of measurement. This property specifies the units MATLAB uses to interpret the `Extent` and `Position` properties. All units are measured from the lower left corner of the axes plotbox.

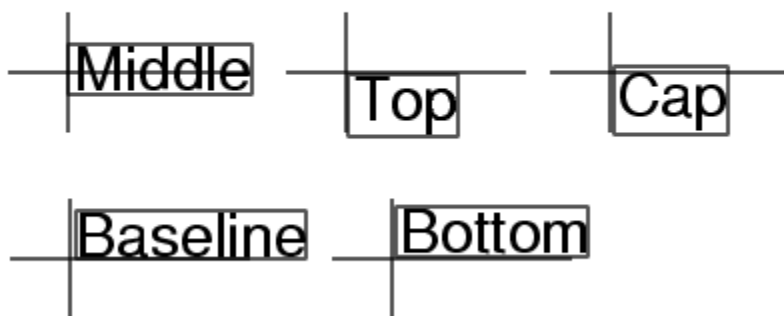
- Normalized units map the lower left corner of the rectangle defined by the axes to (0,0) and the upper right corner to (1.0,1.0).
- pixels, inches, centimeters, and points are absolute units (1 point = $1/72$ inch).
- data refers to the data units of the parent axes.

- **VerticalAlignment** : top | cap | {middle} | baseline | bottom

Vertical alignment of text. This property specifies the vertical justification of the text string. It determines where MATLAB places the string with regard to the value of the `Position` property. The possible values mean

- top -- Place the top of the string's `Extent` rectangle at the specified y-position.
- cap -- Place the string so that the top of a capital letter is at the specified y-position.
- middle -- Place the middle of the string at specified y-position.
- baseline -- Place font baseline at the specified y-position.
- bottom -- Place the bottom of the string's `Extent` rectangle at the specified y-position.

Text VerticalAlignment property viewed with the HorizontalAlignment property set to left (the default).



VIII) Propriétés du set:

- `set(0, 'DefaulttextProperty', PropertyValue...)`
- `set(gcf, 'DefaulttextProperty', PropertyValue...)`
- `set(gca, 'DefaulttextProperty', PropertyValue...)`

Where *Property* is the name of the text property and *PropertyValue* is the value you are specifying. Use [set](#) and [get](#) to access text properties.

Property List

The following table lists all text properties and provides a brief description of each. The property name links take you to an expanded description of the properties.

Property Name	Property Description	Property Value
Defining the character string		
Editing	Enable or disable editing mode.	Values: <code>on</code> , <code>off</code> Default: <code>off</code>
Interpreter	Enable or disable TeX interpretation	Values: <code>tex</code> , <code>none</code> Default: <code>tex</code>
String	The character string (including list of TeX character sequences)	Value: character string
Positioning the character string		
Extent	Position and size of text object	Values: [<code>left</code> , <code>bottom</code> , <code>width</code> , <code>height</code>]
HorizontalAlignment	Horizontal alignment of text string	Values: <code>left</code> , <code>center</code> , <code>right</code> Default: <code>left</code>
Position	Position of text <code>Extent</code> rectangle	Values: [<code>x</code> , <code>y</code> , <code>z</code>] coordinates Default: <code>[]</code> empty matrix
Rotation	Orientation of text object	Values: scalar (degrees) Default: <code>0</code>
Units	Units for <code>Extent</code> and <code>Position</code> properties	Values: <code>pixels</code> , <code>normalized</code> , <code>inches</code> ,

		centimeters, points, data Default: data
VerticalAlignment	Vertical alignment of text string	Values: top, cap, middle, baseline, bottom Default: middle
Text Bounding Box		
BackgroundColor	Color of text extent rectangle	Values: ColorSpec Default: none
EdgeColor	Color of edge drawn around text extent rectangle	Values: ColorSpec Default: none
LineWidth	Width of the line (in points) use to draw the box drawn around text extent rectangle	Values: scalar (points) Default: 0.5
LineStyle	Style of the line use to draw the box drawn around text extent rectangle	Values: -, --, :, -., none Default: -
Margin	Distance in pixels from the text extent to the edge of the box enclosing the text.	Values: scalar (pixels) Default: 2
Specifying the Font		
FontAngle	Select italic-style font	Values: normal, italic, oblique Default: normal
FontName	Select font family	Values: a font supported by your system or the string FixedWidth Default: Helvetica
FontSize	Size of font	Values: size in FontUnits Default: 10 points
FontUnits	Units for FontSize property	Values: points, normalized, inches, centimeters, pixels Default: points

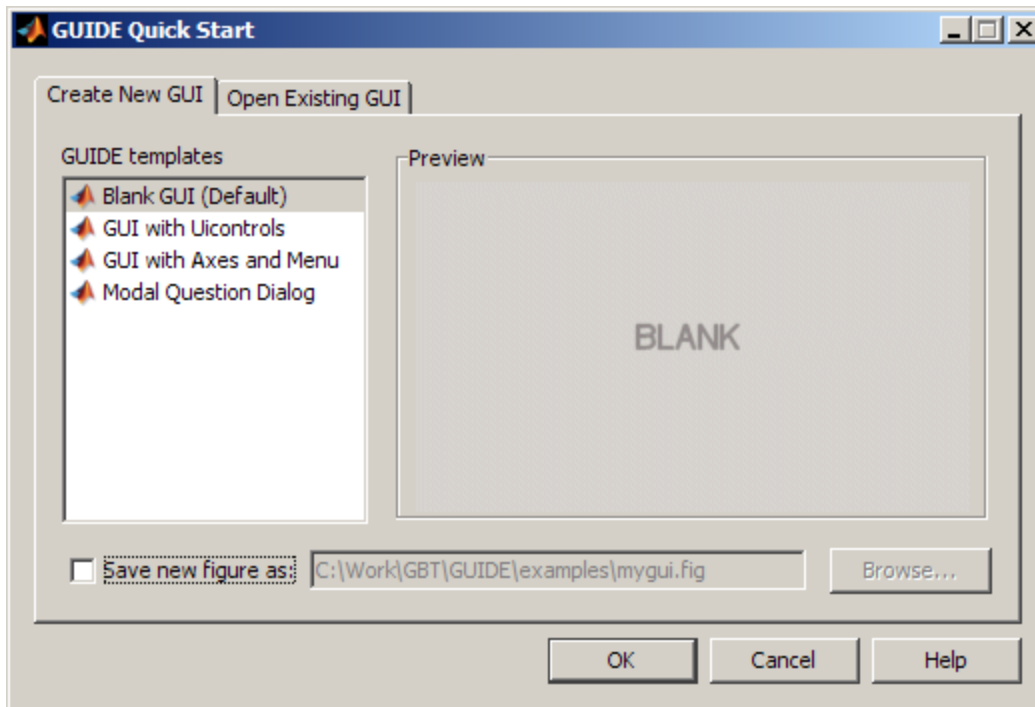
FontWeight	Weight of text characters	Values: light, normal, demi, bold Default: normal
Controlling the Appearance		
Clipping	Clipping to axes rectangle	Values: on, off Default: on
EraseMode	Method of drawing and erasing the text (useful for animation)	Values: normal, none, xor, background Default: normal
SelectionHighlight	Highlight text when selected (Selected property set to on)	Values: on, off Default: on
Visible	Make the text visible or invisible	Values: on, off Default: on
Color	Color of the text	ColorSpec
Controlling Access to Text Objects		
HandleVisibility	Determines if and when the the text's handle is visible to other functions	Values: on, callback, off Default: on
HitTest	Determines if the text can become the current object (see the figure CurrentObject property)	Values: on, off Default: on
General Information About Text Objects		
Children	Text objects have no children	Values: [] (empty matrix)
Parent	The parent of a text object is always an axes object	Value: axes handle
Selected	Indicate whether the text is in a "selected" state.	Values: on, off Default: off
Tag	User-specified label	Value: any string Default: '' (empty string)

Type	The type of graphics object (read only)	Value: the string 'text'
UserData	User-specified data	Values: any matrix Default: [] (empty matrix)
Controlling Callback Routine Execution		
BusyAction	Specifies how to handle callback routine interruption	Values: cancel, queue Default: queue
ButtonDownFcn	Defines a callback routine that executes when a mouse button is pressed on over the text	Values: string or function handle Default: '' (empty string)
CreateFcn	Defines a callback routine that executes when an text is created	Values: string or function handle Default: '' (empty string)
DeleteFcn	Defines a callback routine that executes when the text is deleted (via <code>close</code> or <code>delete</code>)	Values: string or function handle Default: '' (empty string)
Interruptible	Determines if callback routine can be interrupted	Values: on, off Default: on (can be interrupted)
UIContextMenu	Associates a context menu with the text	Values: handle of a <code>uicontextmenu</code>

Chapitre 8 : GUI avec GUIDE

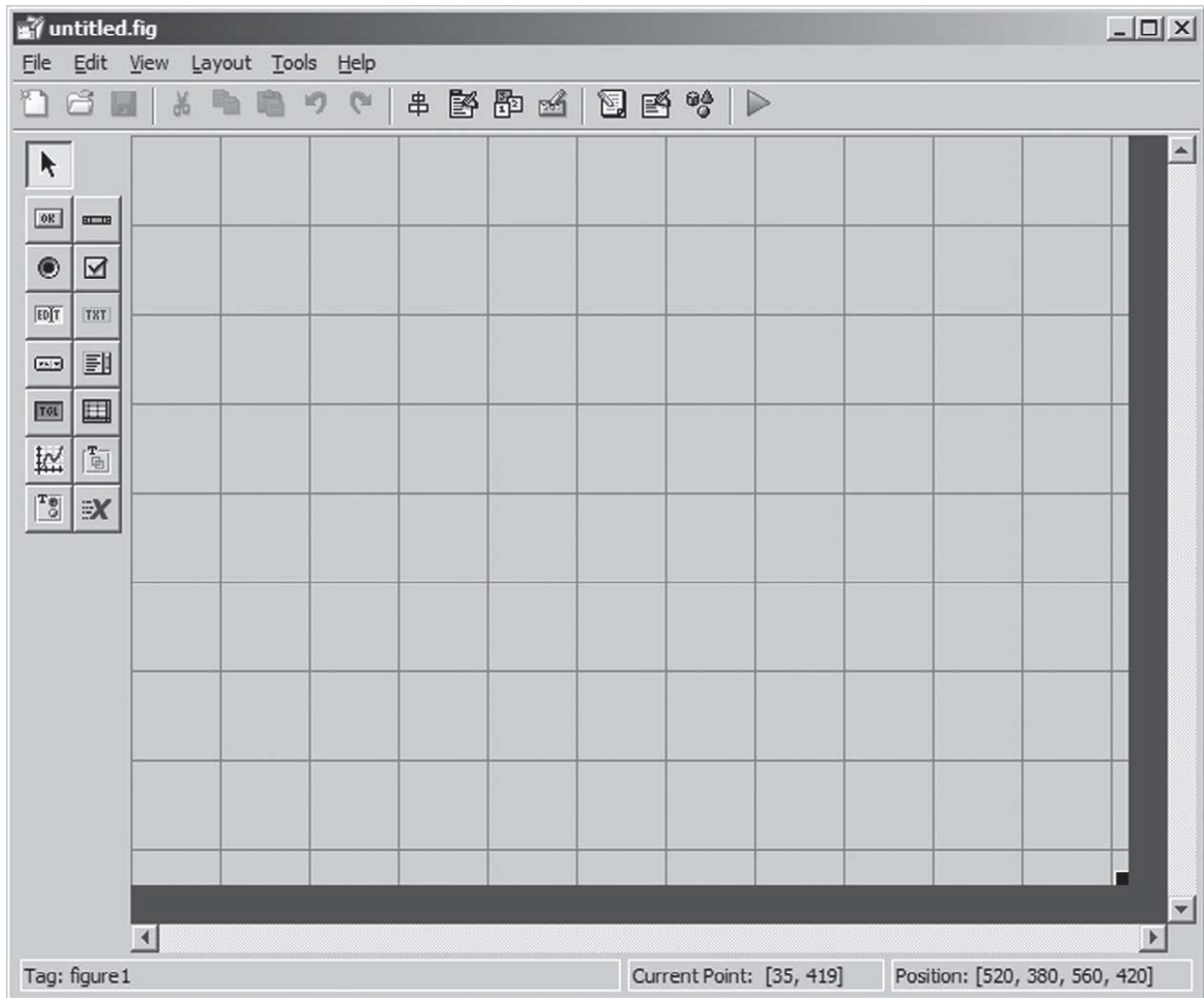
Faite guide au terminal de MATLAB.

Puis la fenêtre suivante :




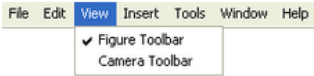



Puis choisir « Blank GUI »

Ce qui devient alors :



Les possibilités :

Nom	Désignation MATLAB	Aperçu
Case à cocher	checkbox	<input checked="" type="checkbox"/> Choix <input type="checkbox"/> Choix
Zone de texte éditable	edit	<input type="text" value="Saisie"/>
Cadre	frame	<div style="border: 1px solid black; width: 50px; height: 20px;"></div>
Liste	listbox	<div style="border: 1px solid gray; padding: 2px;"> item 1 item 2 item 3 item 4 </div>
Menu déroulant	popupmenu	<div style="border: 1px solid gray; padding: 2px;"> item 3 item 3 item 1 item 2 item 3 item 4 </div>
Bouton	pushbutton	<input type="button" value="OK"/>
Bouton Radio	radiobutton	<input checked="" type="radio"/> Choix <input type="radio"/> Choix
Barre de défilement	slider	<div style="border: 1px solid gray; padding: 2px;"> <input type="range"/> </div>
Zone de texte non éditable	text	<input type="text" value="Titre"/>
Bouton à 2 états	togglebutton	<input type="checkbox"/> Off <input type="checkbox"/> On

Nom	Désignation MATLAB	Aperçu
Menus contextuel	uicontextmenu	
Menus	uimenu	
Barres d'outils	uitoolbar	
Bouton (barre outils)	uipushtool	
Bouton à 2 états (barre outils)	uitoggletool	

Chapitre 9 : Questions sur GUI

1) Création d'un GUI plus élaboré:

Lancer guide dans le terminal.

2) Que signifie handles?

Handles : permettent de les repérer dans l'interface; pour envisager par exemple une modification dynamique (grisé d'un bouton provisoirement non utilisable, changement du texte d'un bouton, modification d'une liste de choix...)

3) Que signifient les propriétés dans une GUI?

Propriétés : des objets (couleur, disposition, taille, variable associée)

4) Que signifient les propriétés dans une callback?

décrites en ligne de commande Matlab

5) Qu'est-ce qui signifie l'abréviation GUIDE?

GUIDE = Graphical User Interface Development Environment

6) Que signifie la commande suivante fig1 = figure?

Le paramètre fig1 est le handle de la fenêtre, c'est à dire le numéro de repère de la fenêtre attribué par Matlab à sa création. Il est possible d'appliquer des fonctions sur cette fenêtre (redimensionnement, ajout de menus, boutons, ...) en précisant dans les fonctions le handle

Auquel elle s'applique. La fenêtre active à un instant donné a pour handle implicite gcf.

De façon générale, tout objet graphique se voit attribué un handle ; ce handle sert de référence à cet objet dans l'application.

Il est possible de faire un `get(fig1)`. Il est possible de lire les principales propriétés comme le titre, la position et la dimension dans l'écran, la couleur de fond, la présence et le type de menus, le redimensionnement...

La syntaxe pour lire chaque propriété est la suivante par exemple:

```
valeur_propriete = get( fig1, 'Resize' )
```

Pour modifier une propriété, il faut simplement faire `set(fig1, 'nom_propriété' , valeur_propriété)`:

Par exemple: `set(fig1 , 'Name' , 'Demo GUI' , 'NumberTitle' , 'off');`

La taille et la position de la fenêtre (ou d'un objet) se fixent par modification de sa propriété ou contrôle "position", comprenant les coordonnées (Xor,Yor) du coin inférieur gauche et ses dimensions (Xfen,Yfen):

```
set( fig1 , 'position' , [ 10 , 10 , 300 , 200 ])
```

L'ensemble des propriétés modifiable d'un objet est donné par `set(handle_objet)`. La liste s'affiche avec les valeurs possibles pour les différentes propriétés; les valeurs par défaut sont signalées par des crochets {}.

Tout objet graphique créé pourra être supprimé par :

```
delete (handle_objet)
```

7) Comment insérer un élément avec uicontrol?

L'insertion d'un objet dans une fenêtre se fait par la fonction "uicontrol" , dont le premier paramètre est le handle de la figure de référence. Le deuxième paramètre précise le "style" ou type d'objet à insérer.

Faisons un petit exemple, voici le code:

```
text1 = uicontrol( fig1 , 'style' , 'text' , 'position' , [100,150,170,30] , 'string' , 'Bonjour' , 'fontsize' , 15 )
```

Toutes les propriétés de cet objet peuvent être modifiées par la commande "set". Par exemple, le texte en lui-même ('string') étant une propriété, il peut être modifié:

```
set( text1 , 'string' , 'Au revoir' );
```

Autre exemple: insertion d'un bouton radio

```
radio1 = uicontrol( fig1 , 'style' , 'radio' , 'String' , 'Option1' , 'Position' , [30,30,60,30] )
```

8) Les principaux de l'interaction avec la souris?

- La fonctionnalité la plus courante est la modification de la valeur associée à l'objet (si elle existe): pour les objets destinés à faire une saisie (case à cocher, curseur, champ de saisie, choix de liste...), Matlab gère automatiquement la valeur associée. Cette valeur est récupérée par toute partie de l'application par la fonction "get":

```
valeur = get (handle_objet , 'value');
```

- La deuxième interaction courante est une action déclenchée par le "clic" souris sur l'objet (appuyé puis relâché): la fonction associée est décrite dans la propriété "callback" de l'objet. Cette fonction peut être une instruction de base Matlab ou une fonction définie par l'utilisateur (stocké dans le fichier .m).

```
set( radio1 , 'callback' , 'get( radio1 , 'value' ) ' );
```

- Certains objets n'ont pas de callback (cas des figures) mais possèdent d'autres actions associées à la souris. Leur emploi est identique au callback classique.

Les principaux sont les suivants:

WindowButtonUpFcn WindowButtonDownFcn WindowButtonMotionFcn

Exemple : récupération des coordonnées en pixels de la souris au clic

```
fig1 = figure ;
```

```
set( fig1 , 'WindowButtonDownFcn' , 'get( fig1 , 'CurrentPoint' ) ' );
```

Si on désire obtenir des coordonnées dans l'espace de mesure des axes d'un graphique, plutôt qu'en pixels de la figure, il faut faire référence aux axes (gca) dans la fonction de clic:

```
plot( 20 , 20 , ' r+ ' ) % tracé d'une croix rouge au centre
```

```
set((gcf , 'WindowButtonDownFcn' , 'get( gca , 'CurrentPoint' ) ' )
```

- Certains objets possèdent une fonction callback et une fonction associée au clic souris (par exemple : ButtonDownFcn)

9) Les principaux Objets Graphiques?

- Bouton poussoir

Un bouton poussoir se crée par :

```
bp1= uicontrol ( fig1 , 'style' , 'push' , 'position' , [10 100 60 30 ] , ...  
'string' , 'Début' , 'callback' , 'plot(T,X)' )
```

Lorsqu'on clique sur le bouton poussoir, il provoque l'exécution de la fonction indiquée dans le 'callback'. Cette fonction peut-être une instruction de la base Matlab ou une liste d'instruction, ce qui évite d'écrire une multitude de petites fonctions exécutées par les callbacks.

Un bouton-poussoir s'inactive par la commande :

```
set(bp1 , 'enable' , 'off' )
```

- Menus

Généralement, les menus de la fenêtre d'application ne sont pas les menus standards Un menu est un titre complété par une liste de sous-menu. Les actions (callbacks) sont généralement lancés à partir des sous-menus. L'ajout de menus spécifique se fait par :

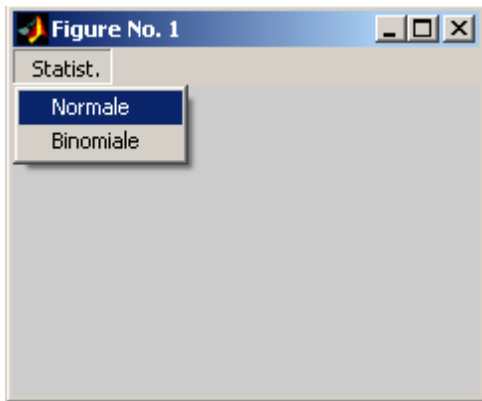
```
menu1 = uimenu( fig1 , 'label' , ' Statist.' );
```

Un sous-menu est un élément du menu principal, donc de l'entité père. Il est donc déclaré car le menu du menu principal.

```
smenu1 = uimenu( menu1 , 'label' , 'Normale' , 'callback' , 'methode_normale' )  
smenu2 = uimenu( menu1 , 'label' , 'Binomiale' , 'callback' , 'methode_binomiale' );
```

Pour enlever les menus standards de la fenêtre, il faut fixer la propriété "Menubar" à la valeur par défaut menubar:

```
set(fig1,'menubar',menubar);
```



➤ Ascenseur ou slider

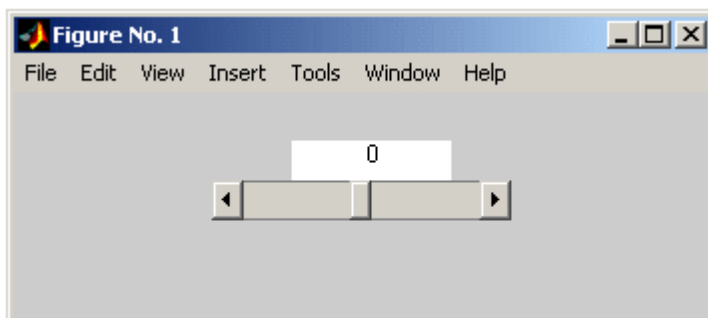
L'ascenseur a pour objectif de fixer la valeur d'un paramètre entre deux bornes fixées. La valeur prise par la variable est représentée par la position du curseur.

```
slid1=uicontrol(fig1,'style','slider','position', [100,50,150,20] , 'Min' , -1 , 'Max' , 1 , ...
'callback' , 'val_variable = get(slid1 , "value" )');
```

Les textes (variable affectée, valeurs...) ne sont pas définis par le slider. Il faut le compléter par des éléments textes convenablement placés et paramétrés; leur valeur est à modifier par la callback du slider.

Exemple:

```
fig1=figure;
texte1=uicontrol(fig1,'Style','text','String','0','Position', [140,70,80,20],'BackgroundColor','w');
slid1=uicontrol(fig1,'style','slider','position', [100,50,150,20] , 'Min' , -1 , 'Max' , 1 , ...
'callback' , 'set(texte1,"String", get(slid1 , "value" ))');
```



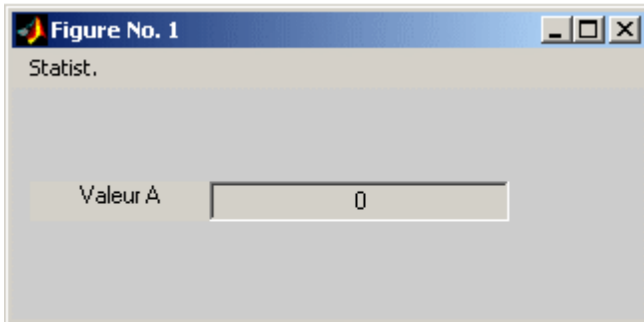
➤ Texte Editable

Permet à l'utilisateur de saisir une valeur. C'est une fonction importante.

```
Text1 = uicontrol ( fig1 , 'style' , 'edit' , 'position' , [100,50,150,20] , 'Max' , 1 , 'string' , '0' );
```

Généralement, il faut associer un texte fixe pour préciser le rôle de la fenêtre de saisie à l'utilisateur.

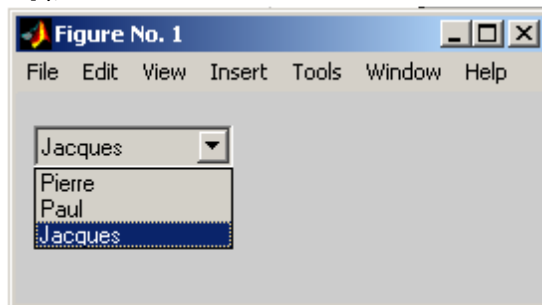
```
uicontrol ( fig1 , 'style' , 'texte' , 'position' , [10,50,90,20] , 'string' , 'Valeur A' );
```



➤ Liste de choix

La liste de choix ou pop-up menu permet de sélectionner une valeur parmi une liste. Généralement, cette valeur est un texte. La valeur retournée lors du choix (paramètre 'Value') est le numéro de ligne du choix.

```
choix1 = uicontrol ((gcf , 'Style' , 'popup' , 'String' , 'Pierre|Paul|Jacques' , 'Position' , [10 10 100 80] );
```



➤ Bouton Radio

Le bouton Radio permet de fixer un paramètre binaire (0 ou 1), représentant souvent un choix ou une option dans une application.

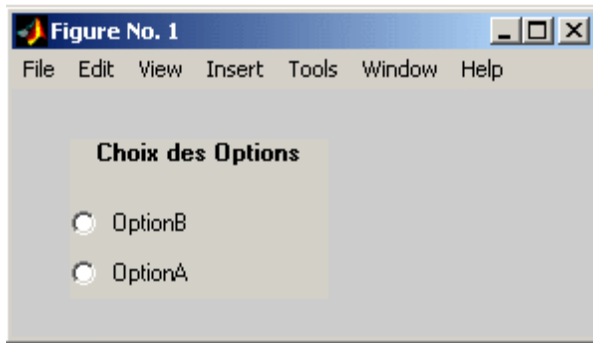
```
fig1 = figure ;
```

```
radio1 = uicontrol( fig1 , 'style' , 'Radio' , 'Position' , [ 30 20 130 25 ] , 'String' , ' OptionA ' );
```

```
radio2 = uicontrol( fig1 , 'style' , 'Radio' , 'Position' , [ 30 45 130 25 ] , 'String' , ' OptionB ' );
```

```
uicontrol( fig1 , 'style' , 'Text' , 'Position' , [ 30 70 130 30 ] , 'String' , ...
```

```
' Choix des Options ' , 'FontWeight' , 'bold' );
```

- **Cadre**
Le cadre permet de dessiner un rectangle de présentation (par exemple regroupement de diverses entités graphiques dans un rectangle).

Le cadre se déclare par :

```
cadre1 = uicontrol ( fig1 , 'style' , 'frame' , 'position' , [ posX ,posY,tailleX,tailleY])
```

- **Graphiques**
Les graphiques se dessinent dans une partie de la fenêtre définie par la fonction 'subplot', dont les paramètres sont différents de l'emploi classique (division de la fenêtre en sous-fenêtres de taille égale)

```
subplot( 'Position' , [ Xpos Ypos Xtaille Ytaille])
```

Exemple :

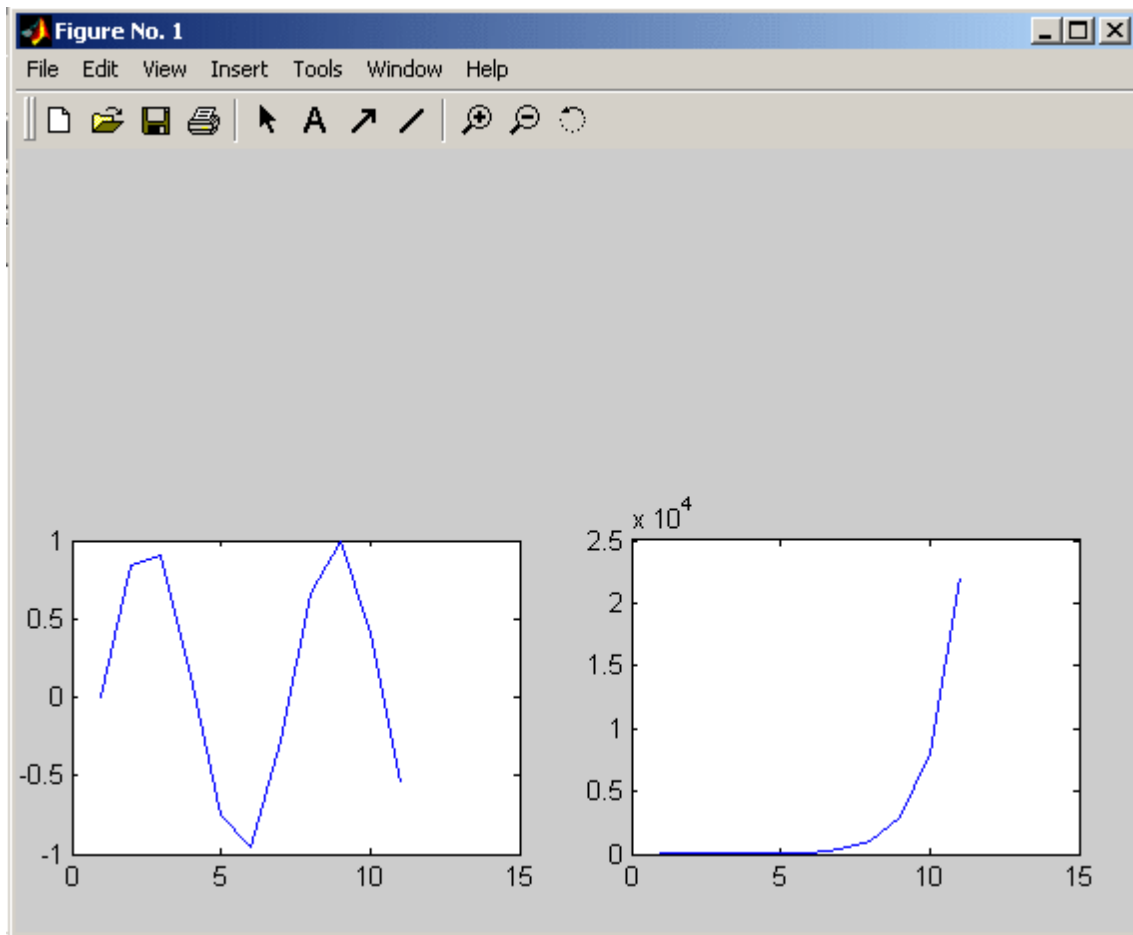
```
fig1 = figure ;
```

```
z1 = subplot ( 'Position' , [ .05 .1 .4 .4 ] ) ;
```

```
plot ( sin( 0: 10))
```

```
z2 = subplot ( 'Position' , [ .55 .1 .4 .4 ] ) ;
```

```
plot ( exp( 0 : 10))
```



10) Quelques propriétés des objets d'interface

Tag :	'Text1', 'Edit2', 'Slider1', 'Axes1'
String :	'LireFT+Step', ou 'Terminer', ou 'Boucle fermée'
Callback :	'close(gcf)' ou 'grid on', ...
ButtonDownFcn	'animate start'
WindowButtonDownFcn, WindowButtonMotionFcn	
ForegroundColor, BackGroundColor, Color: [Rouge Vert Bleu]	
Value :	1 : Checkbox cochée, 0 : non cochée

FontAngle	'italic' (text)
FontName	'Brush Script'
FontSize	16
FontWeight	'bold'
Max, Min	50, et 1 (slider)
SliderStep	[0.01 0.1] (slider)
Position	[100 100 200 400] soit xBG, yBG, Lx, Ly
Name	'Mon premier interface'
Pointer	'arrow', ou 'fullcrosshair' (figure)
CurrentPoint	Currpt=get(gca,'currrentpoint') → x et y souris
NextPlot	'add'

11) Comment mettre un texte dans une label?

Par exemple avec l'action d'un bouton et ce label aura pour nom (Tag¹) textStatus.

On voudra appuyer sur un bouton puis afficher le texte suivant "Bouton appuyer".

Le code est le suivant :

```
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)

set(handles.textStatus, 'String', 'Bouton appuyer')
```

Si on utilise maintenant un "Toggle Button" (donne une réponse), on souhaite afficher quand le Toggle Button relâcher l'instruction suivante 'Toggle down) sinon 'Toggle up'.

```
function togglebutton1_Callback(hObject, eventdata, handles)
```

¹ étiquette

```
isDown = get(hObject,'Value');
```

```
if isDown
```

```
    set(handles.textStatus, 'string', 'Toggle down')
```

```
else
```

```
    set(handles.textStatus, 'string', 'Toggle up')
```

```
end
```

12) Comment mettre un texte dans un pop-up menu?

Je souhaite sélectionner les textes du pop-up menu et de changer le colormap et afficher le changement dans un label nommé "textStatus".

```
function popupmenu1_Callback(hObject, eventdata, handles)
```

```
contents = get(hObject,'String');
```

```
selectedText = contents{get(hObject,'Value')};
```

```
colormapStatus = [selectedText ' colormap'];
```

```
set(handles.textStatus, 'string', colormapStatus);
```

```
colormap(selectedText)
```

13) Comment mettre une image dans un bouton?

Les commandes de l'action sont les suivantes:

```
Img = rand(16,4,3);
```

```
set(handles.togglebutton1, 'CData', img);
```

ou aussi

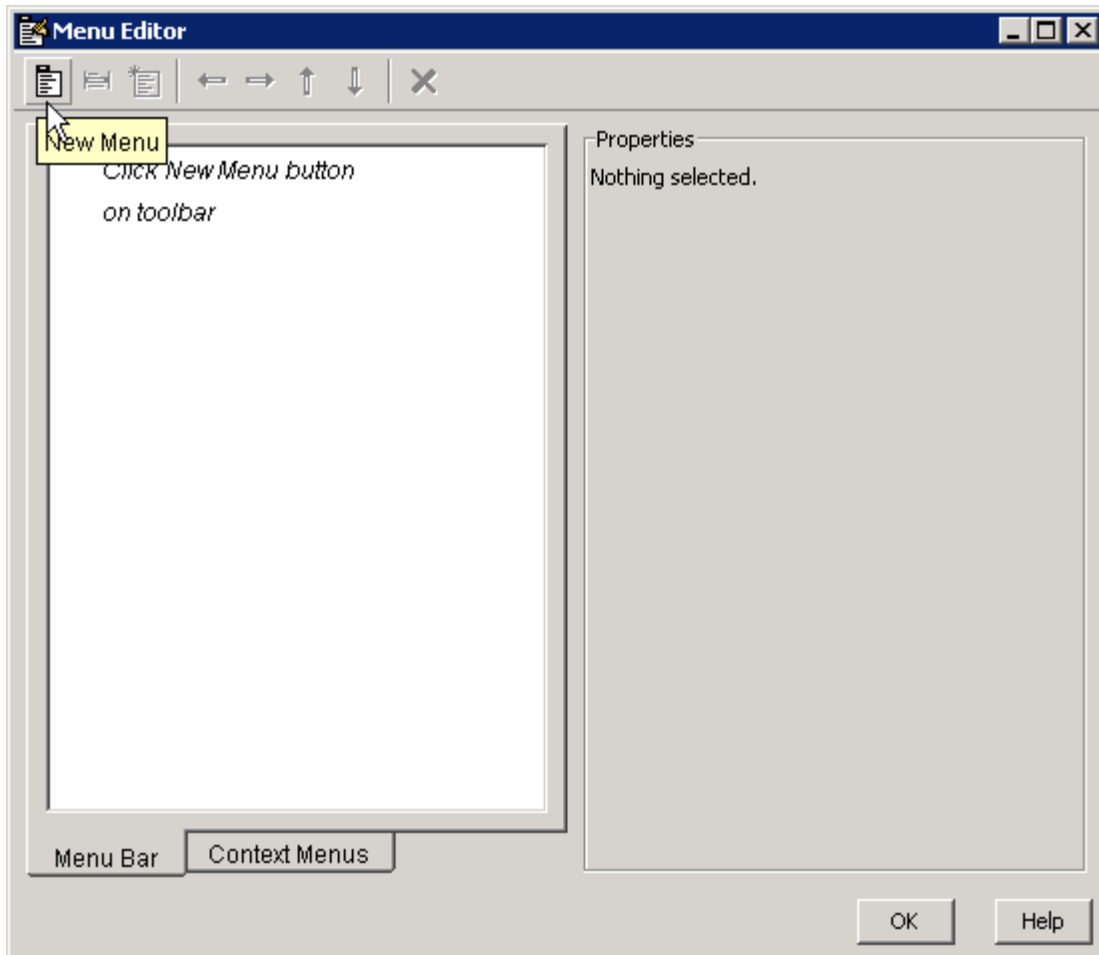
```
Img = imread('nom de l'image avec l'extension');
```

```
set(handles.togglebutton1, 'CData', img);
```

14) Comment créer un menu ou sous-menu?

Aller dans le Menu du GUIDE puis dans TOOLS puis Menu Editor.

Voici la fenêtre affichée :



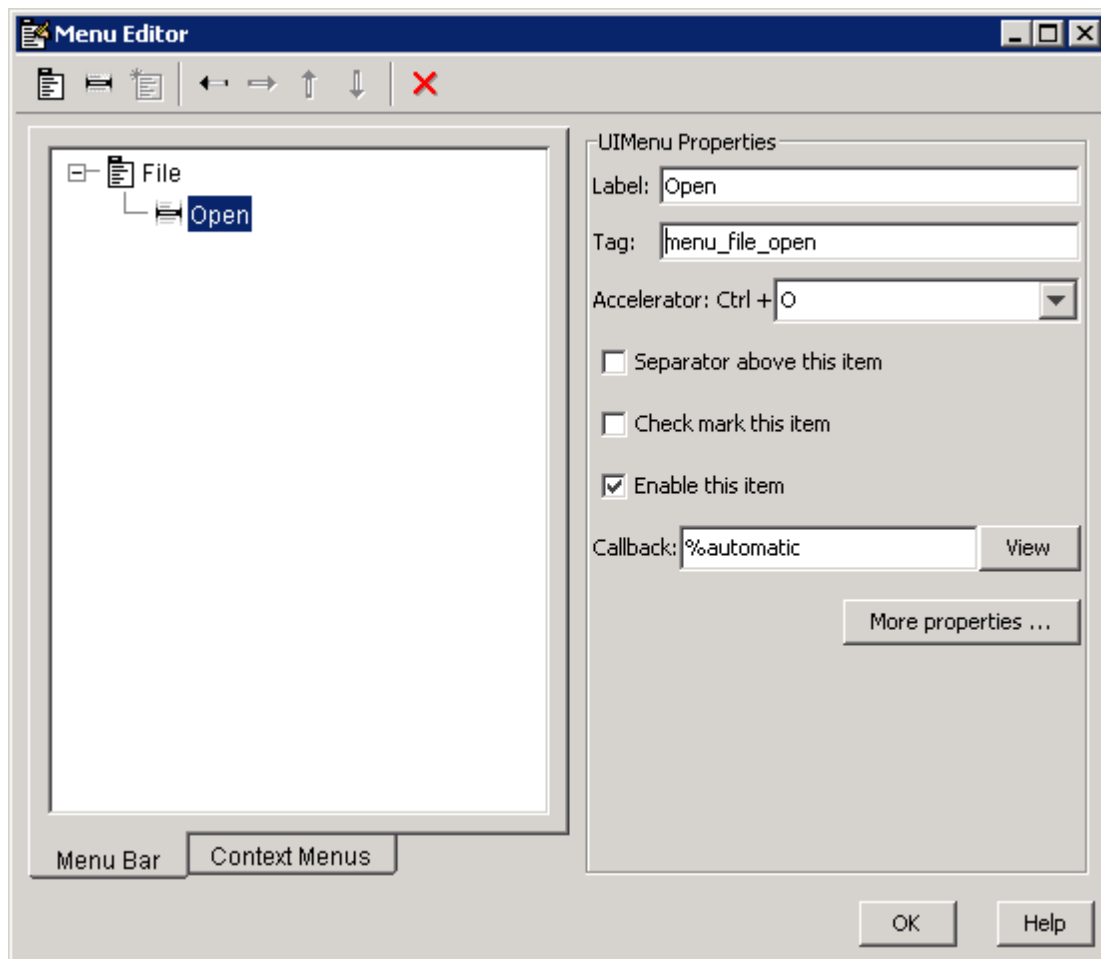
Puis par exemple, faire un menu File est un sous-Menu Open.

On crée l'onglet File appuyer sur New Menu, puis idem pour Open puis appuyer sur la flèche de droite du Menu Editor.

Il est possible de changer le nom du Menu appeler (Label) puis son étiquette² (Tag).

Il est possible de mettre un raccourci, séparer les sous-menus et de mettre une action (Callback).

² On étiquette le nom de l'action pour le Menu



15) Signification de hObject, eventdata et handles:

- **hObject**— Handle of the object, e.g., the GUI component, for which the callback was triggered. For a button group SelectionChangeFcn callback, hObject is the handle of the selected radio button or toggle button.
- **eventdata** — Sequences of events triggered by user actions such as table selections emitted by a component in the form of a MATLAB struct (or an empty matrix for components that do not generate eventdata)
- **handles** — A MATLAB struct that contains the handles of all the objects in the GUI, and may also contain application-defined data.

16) Signification de la fonction mygui_OpeningFcn(hObject, eventdata, handles, varargin):

Cette fonction est généré automatiquement par Matlab.

```
function mygui_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 mygui (see VARARGIN)
% Choose default command line output for mygui
handles.output = hObject;

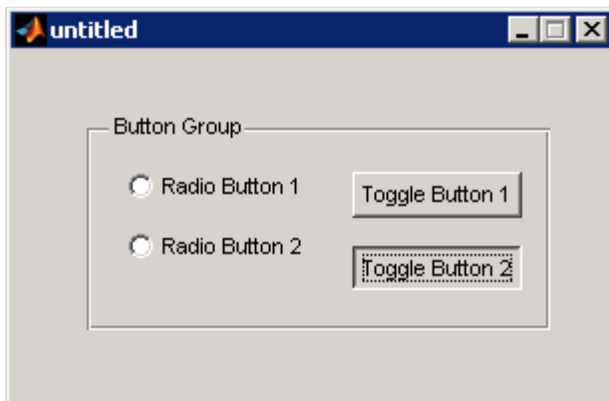
```

```

% Update handles structure
    guidata(hObject, handles)3;

```

17) Comment faire un bouton grouper?



Choisir l'action SelectionChangeFcn, voici le code :

```

function uibuttongroup1_SelectionChangeFcn(hObject,eventdata)
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
    case 'radiobutton1'
% Code for when radiobutton1 is selected.
    case 'radiobutton2'

```

³ Le guidata permet de remettre à jour la structure handles.

```

% Code for when radiobutton2 is selected.
    case 'togglebutton1'
% Code for when togglebutton1 is selected.
    case 'togglebutton2'
% Code for when togglebutton2 is selected.
% Continue with more cases as necessary.
    otherwise
% Code for when there is no match.
end

```

18) Comment faire subplot dans une GUI?

```

a1=subplot(2,1,1,'Parent',handles.uipanel1);
plot(a1,rand(1,10),'r');
a2=subplot(2,1,2,'Parent',handles.uipanel1);
plot(a2,rand(1,10),'b');
axes(a1)
plot(a1, ...)
set(gcf,'CurrentAxes', a1)
line(x,y,z,...)

```

19) Comment rafraîchir une fenêtre?

```

hObject.radius = floor(.9*hObject.radius);
hObject.label = ['Radius = ' num2str(hObject.radius)];
refresh(handles.figure1);

```

20) Boite de dialogue d'ouverture ?

```

[file,path] = uiputfile('animinit.m','Save file name');

```


21) Les données ?

```
function mygui_edittest1(hObject, eventdata, handles)
mystring = get(hObject,'String');
set(hObject,'UserData',mystring)
string = get(handles.edittest1,'UserData');
```

Chapitre 10 : Différentes fonctions utiles pour GUI

Waitbars

```
<handle>=waitbar(0, '<message string>', 'Name', '<title>');
```

Pour iteration,

```
waitbar(progress, <handle>);
```

exemple :

```
N=500;
hwb=waitbar(0, 'Calculating ...', 'Name', 'Time marching');
for k=1:N

% <long calculation goes here>
waitbar(k/N, hwb);
end
close(hwb)
```

File dialogs

Save

```
save myData x y t
```

ou

```
fname='myData';
save(fname, 'y', 'yinit', 'yfinal');
```

Lire

```
load myData % loads all variables from myData.mat
load myData x y name % loads variables x, y, name from myData.mat
```

ou

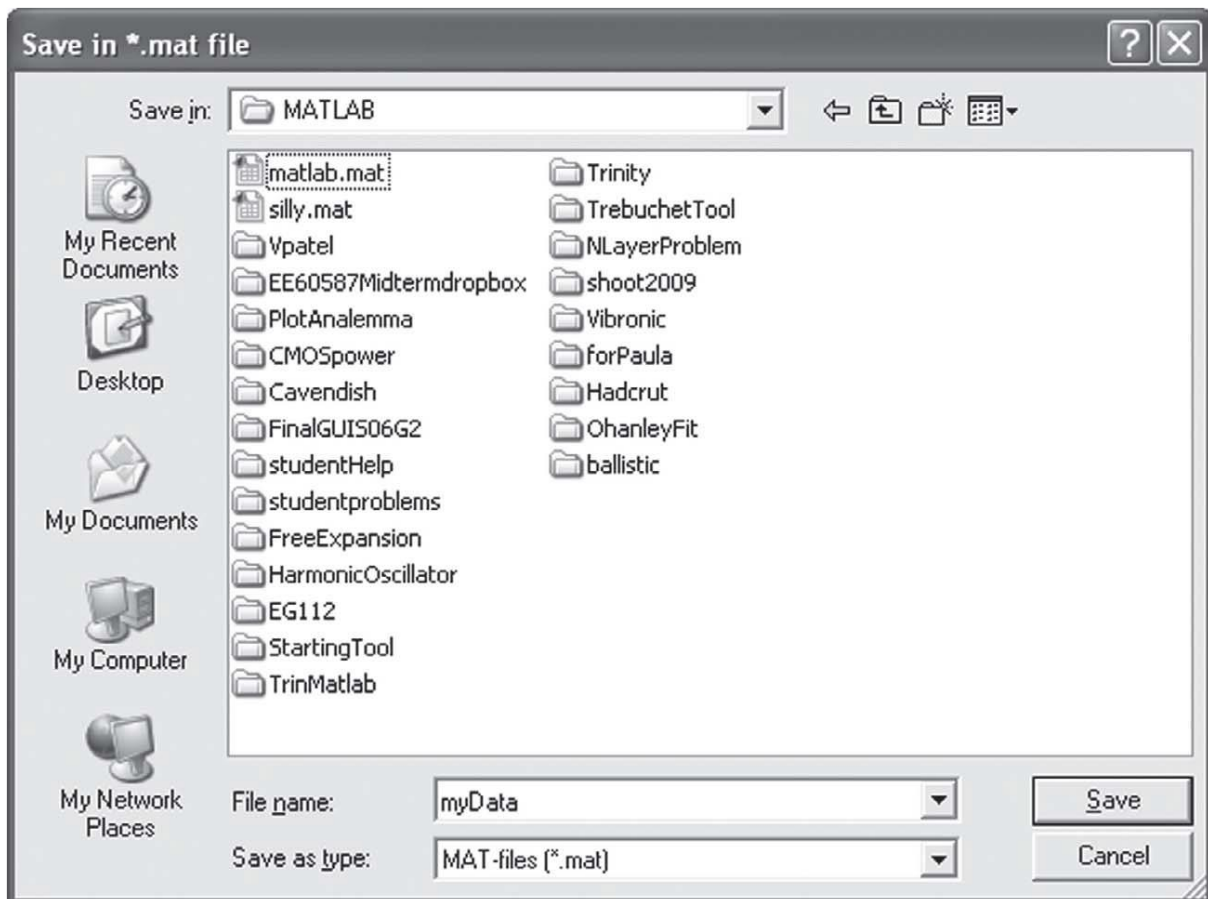
```
theFile='myData';
load(theFile);
load(theFile, 'x', 'y', 'name')
```

uinputFile

```
[FileName, PathName, FilterIndex] = uinputfile(FilterSpec,...
DialogTitle, DefaultName);
```

Ou

```
[fname, pname]=uinputfile('*.*mat', 'Save in .mat file', 'myData');
```

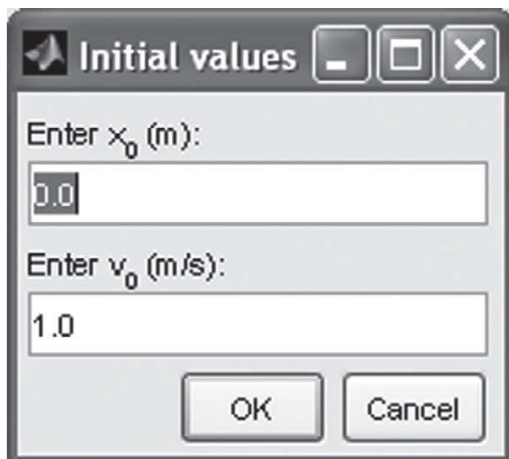


Input dialog

```

prompt={'Enter x_0 (m):',...
'Enter v_0 (m/s):'};
name='Initial values';
numlines=1;
defaultanswer={'0.0', '1.0'};
options.Resize='on';
options.Interpreter='tex';
caStr=inputdlg(prompt,name,numlines,defaultanswer,options);

```



Question dialog

```
choice=questdlg('Do you want to sing the blues?',  
'Important question');  
wantsToSingTheBlues=false;  
switch choice  
case 'Yes'  
disp('Then you gotta pay your dues.');wantsToSingTheBlues=true;  
case 'No'  
disp('Okay, then.');case 'Cancel'  
disp('Cancelled.')end
```

