Apprendre Matlab

Par Dimitri PIANETA

Édition 2015

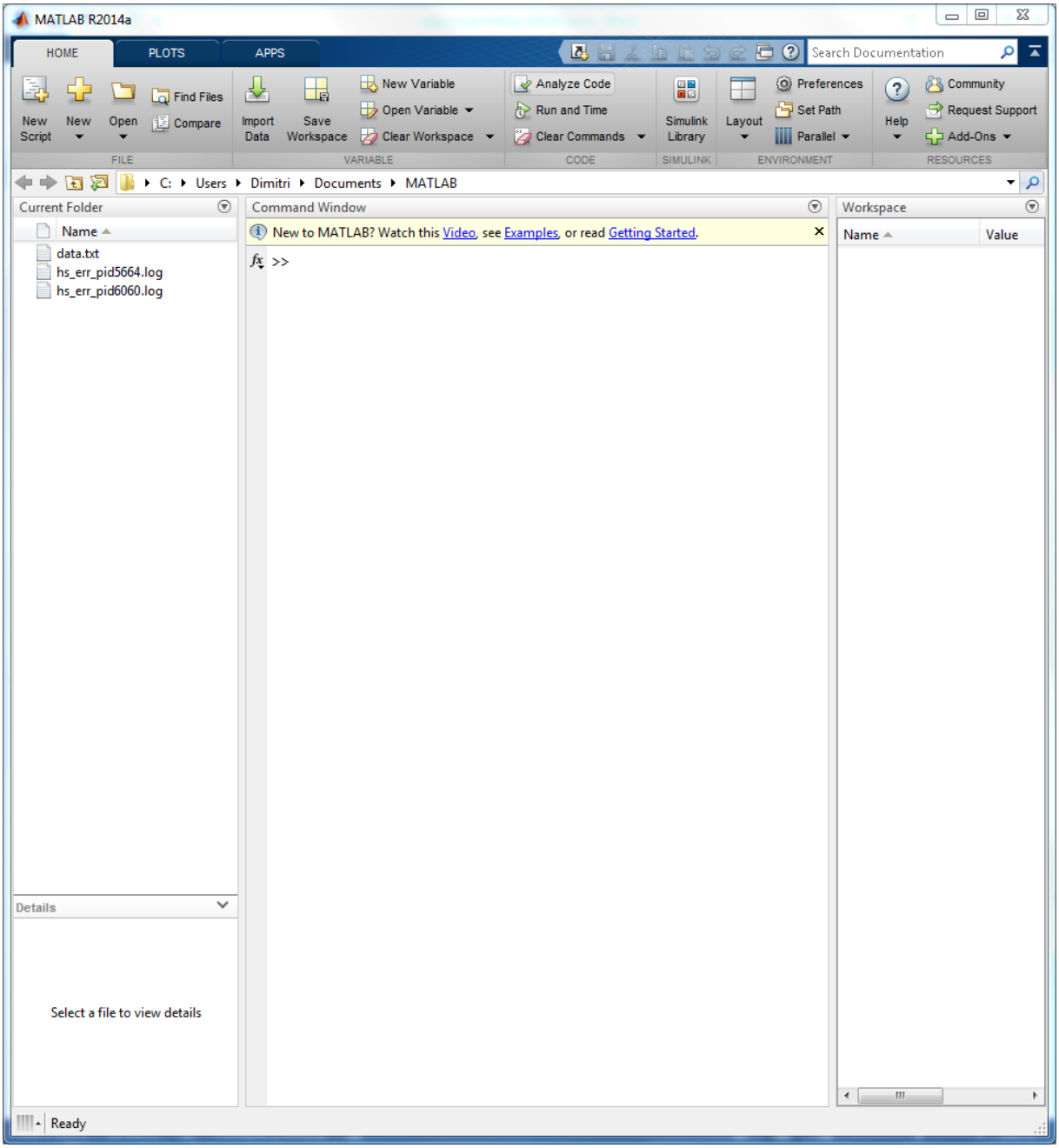
Table des matières

IDE MATLAB....3
NUMERIQUE....7
Chapitre1 : les types....8
Chapitre2 : les fonctions....13
Chapitre3 : Script et fonctions....16
Chapitre 4 : Les tableaux et les boucles itératives....18
Chapitre 5 : Les opérateurs....20
Chapitre 6 : Structures de données....22
Chapitre 7 : Les chaînes de caractères....23
Chapitre 8 : L'algèbre linéaire....24
Chapitre 9 : Polynômes, interpolation, et statistiques de base....25
Chapitre 10 : Entrées et sorties....27
Interface graphique....28
Chapitre 11 : Le plot....29
Chapitre 12 : figure....35
Chapitre 13 : les images....42
Chapitre 14 : Changer le pointer de la souris une figure....44
IHM....45
Chapitre 15 : les bases....46
Chapitre 16 : Approfondissement....48
Chapitre 17 : Poignées graphiques....52
Chapitre 18 : 1er exemple à la main....53
Chapitre 19 : Guide....61
Chapitre 20: Questions sur GUI....62
Chapitre 21 : Différentes fonctions utiles pour GUI....78
Chapitre 22 : Compléments....81
Chapitre 23: Exemple de guide....86
Projet IHM avec GUIDE....93
Signal analyse:....94
Interface Image....100
Calculatrice....112

Dans ce cours de Matlab, je me base sur la version R2014a.

IDE MATLAB

Voici l'interface graphique de Matlab :

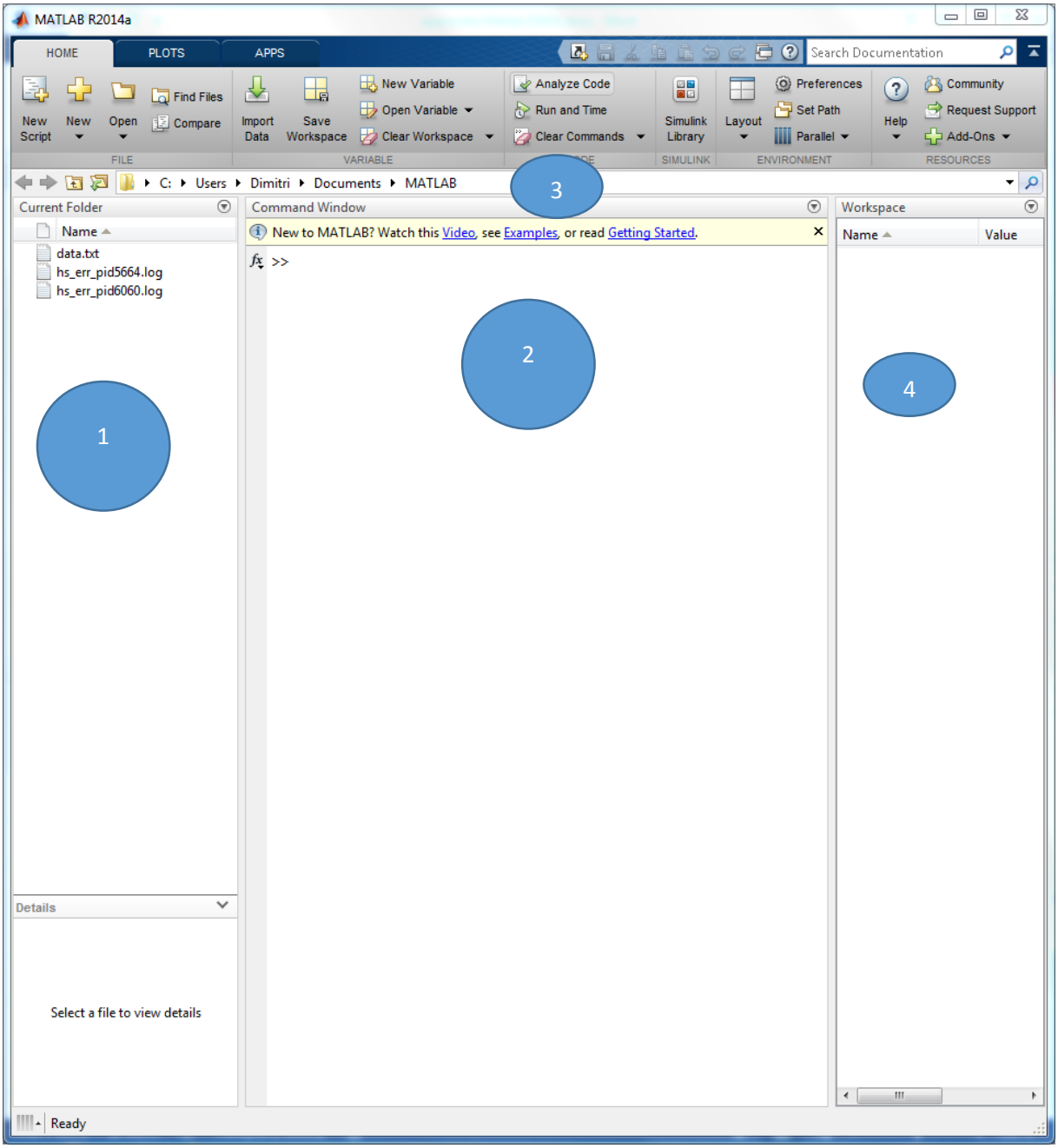


Cette nouvelle version est basée sur les onglets comme une grande partie des logiciels sur Windows.

- ***L'onglet « Home »*** : sont les tâches courantes pour gérer les variables, les scripts, les importations de données, les figures, aides
- ***L'onglet « Plots »*** : sont les types de figures

- **L'onglet « Apps »** : sont les éléments de simulink[1]

Puis l'IDE :



Description :

(1) : Sont la liste des fichiers du répertoire courant (3) pour compiler
(2) : le terminal ou le prompt selon les personnes
(3) : le chemin courant
(4) Les variables

[1] Simulink est basé sur Matlab pour gérer des éléments automatismes.

- 1er Activité : ouvrir editeur matlab

1er solution tapé dans le terminal :

>> edit

2ème solution allez sur « New script » dans l'onglet HOME

- 2ème Activité : regarder l'aide

1er solution tapé dans le terminal :

>> help

Résultat :

matlabhdlcoder\matlabhdlcoder - (No table of contents file)

matlab\testframework - (No table of contents file)

matlabxl\matlabxl - MATLAB Builder EX

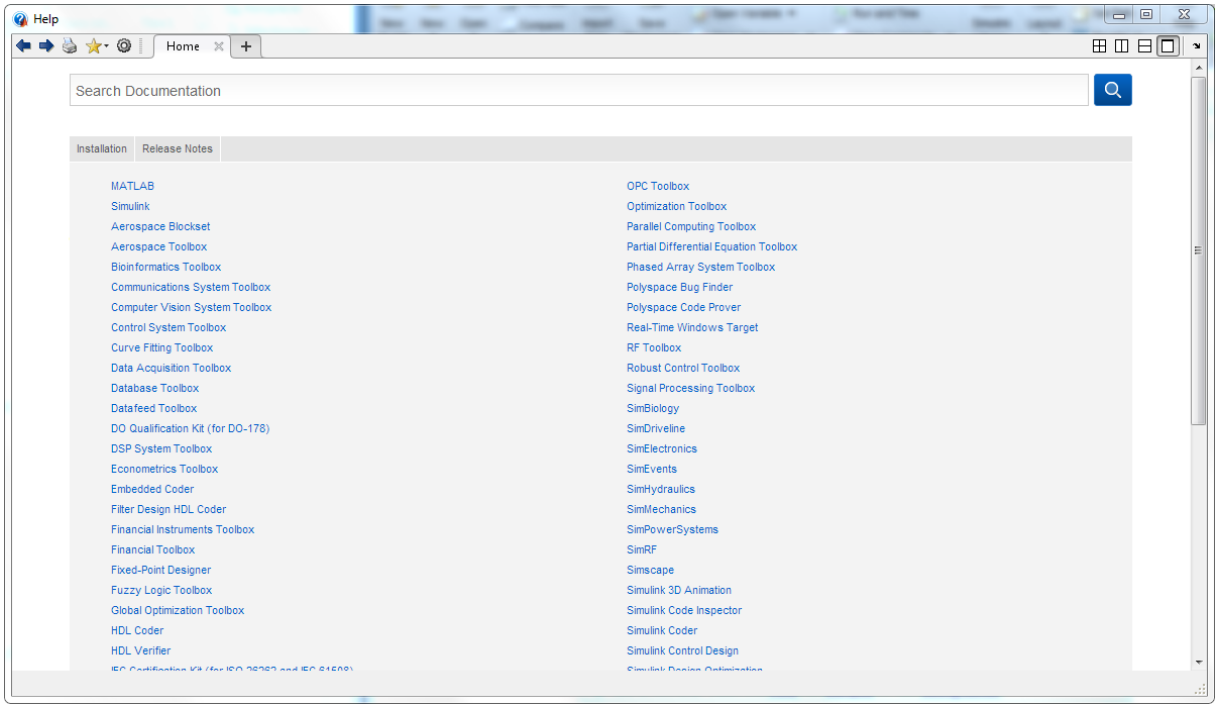
matlab\demos - Examples.

matlab\graph2d - Two dimensional graphs.

matlab\graph3d - Three dimensional graphs.

2ème solution allez sur "Help"

Vous devez avoir comme résultat :



Vous êtes prêts à apprendre MATLAB.

NUMERIQUE

Chapitre1 : les types

a) Nombres réels :

La commande : format

Cette commande permet de retourner le type ici d'entier.

Prenons cette exemple ce rationnel : 1/7

1. **>> 1/7**
2. **ans = 0.1429**

La ligne 1 est l'opération dans le terminal et la ligne 2 est le résultat.

Reprenons notre exemple précédent : 1/7
On va convertir le type par la fonction format.

>> format short
>>ans

Voici le résultat des différents types :

format short donne 0.1429,
format shortE donne 1.4286e− 01,
format shortG donne 0.14286,
format shortEng donne 142.8571e-003
format long donne 0.142857142857143,
format longE donne 1.428571428571428e− 01,
format longG donne 0.142857142857143.
format longEng donne 142.857142857143e-003

b) Nombres complexes :

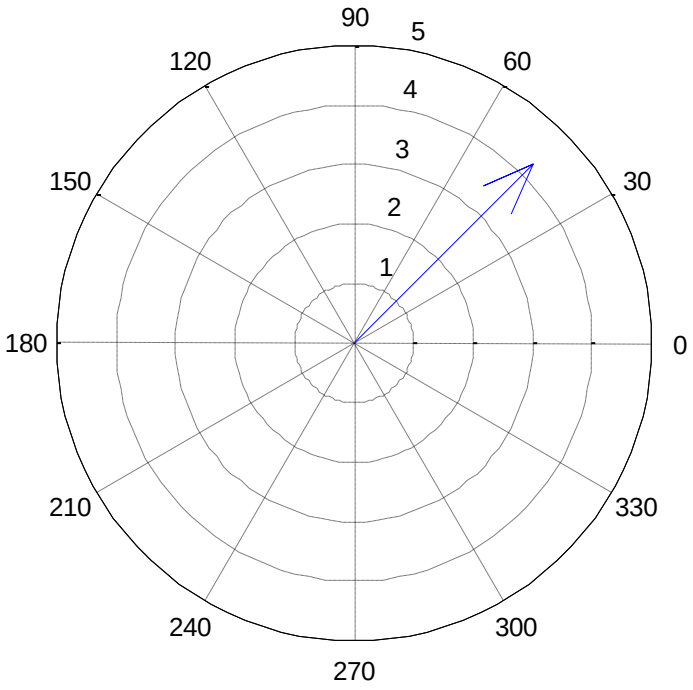
Pour définir un complexe, on peut utiliser la commande complex(x,y) qui est égale à $z=x+iy$ avec $i^2=-1$

abs(z)
angle(z) : argument

On peut écrire que abs(z)*cos(angle(z))+i*sin(angle(z))

La représentation complexe compass(z).

Ex : >> z = 3+i*3 ; compass(z) ;



On peut extraire la partie réelle d'un nombre complexe z avec la commande real(z) et sa partie imaginaire avec imag(z). Enfin, le complexe conjugué $\overline{z} = x - iy$. Ce fait conj(z).

c) Matrices :

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

A =

1 2 3
4 5 6
7 8 9

Remarques :

- Sensible à la casse (a≠A)
- [] création : création
- Espace ou , :changement de colonne
- ; ou Entrée : changement de ligne

>>A = [1 2 3
4 5 6
7 8 9]

Ou A = [1 2 3 ; 4 5 6 ; 7 8 9]

- **Lecture d'un élément dans la matrice :**

>> A(2,1)

Ligne 2 et colonne 1 ce qui nous donne comme résultat 4.

- **Opérations**
  - Additions :

```
>> A=[1 2 3; 4 5 6];
>> B=[7 8 9; 10 11 12];

>>A+B

ans =
8 10 12
14 16 18
```

  - Soustractions :

```
>> A-B

ans =

   -6   -6   -6
   -6   -6   -6
```

  - Multiplications :

```
>>C=[13 14; 15 16; 17 18];
>>A*C

ans =

   94   100
  229   244
```

- **Inversion de la matrice**

```
>> inv(A)
```

- **Déterminant de la matrice**

```
>> det(A)
```

- Diagonalisation

```
>>diag(A)
```

tril(A) et triu(A) permettent d'extraire les parties triangulaires supérieure et inférieure d'une matrice A de dimension n.

d) Vecteurs :

On appelle vecteur :

```
>> V = [3 1]
>> v(1)
ans = 3
>> 1 :5
ans = 1 2 3 4 5

>> 1 :2 :10
ans = 1 3 5 7 9
>> 2 :-1 :1
ans : 2 1
```

Les opérations possibles de faire :

- La transposée :

On met ‘ au vecteur.

- Produit scalaire

dot(v,w)

ex : A = [4 -1 2];
B = [2 -2 -1];
>> C = dot(A,B)
C =

8

Calcul de la fonction dot :

C = A(1)*B(1) + A(2)*B(2) + A(3)*B(3)
C = 8 + 2-2

- Norme euclidienne d’un vecteur v

>> norm(v)

Formule :

$$\|v\| = \sqrt{(v,v)} = \sqrt{\sum_{k=1}^{n} v_k^2}$$

- Le produit vectoriel :

>> cross(v,w)

- Des fonctions de réaction de vecteur :

ones(n,1)

ex :

```
>> ones(2,1)
```

ans =

1
1

- Afficher des vecteurs

Quiver dans domaine 2D réels et quiver3 dans un domaine 3D réels.

quiver(x,y,u,v) est quiver ou affichage de la vélocité

```
>> [x,y] = meshgrid(0:0.2:2,0:0.2:2);
u = cos(x).*y;
v = sin(x).*y;
figure
quiver(x,y,u,v)
```

quiver3(x,y,z,u,v,w) est le 3D quiver ou affichage de la vélocité

a) Données manquantes :

Les données manquantes sont désignées conventionnellement par **NaN**.

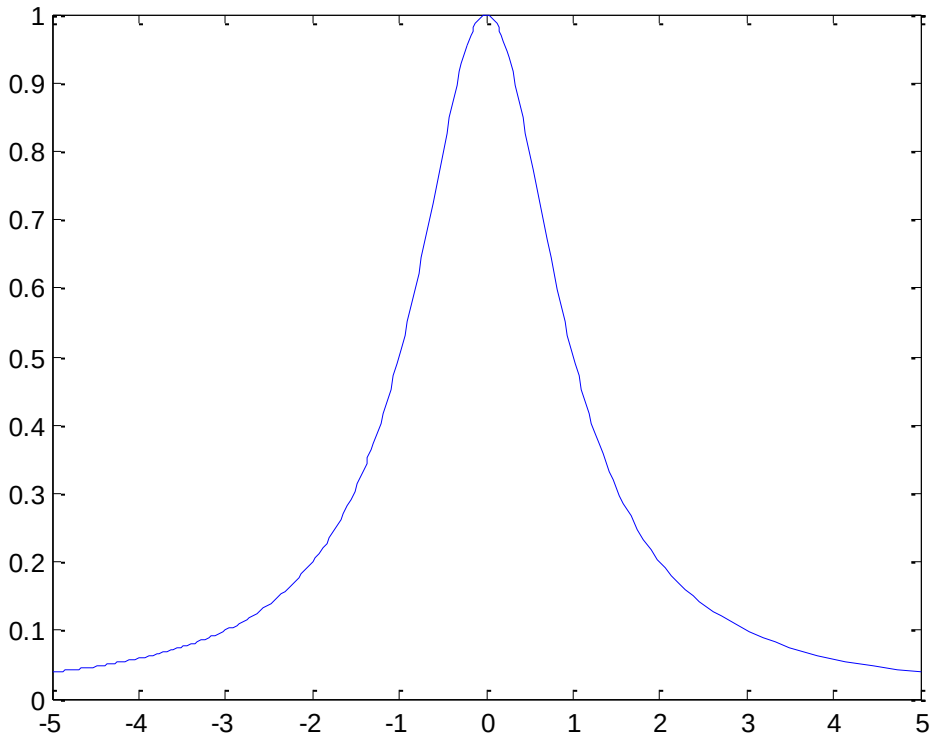
Pour lever cette vérifier, on utilise la méthode **isnam**

Chapitre2 : les fonctions

1) Fonctions réelles

La commande fplot(fun,lims) trace le graphe de la fonction fun sur l'intervalle]lims(1), lims(2)[.

```
>> fun ='1/(1+x^2)'; lims=[-5,5]; fplot(fun,lims);
```



On peut affiner l'affichage par :

```
>> fplot(fun,lims,tol,n,LineSpec)
```

Où

tol indique la tolérance souhaitée et le paramètre n (>=1) assure que la function sera tracée avec un minimum de n+1 points.

LineSpec spécifie le type de ligne ou la couleur (par exemple, LineSpec='-' pour une ligne en traits discontinus, LineSpec='r- -', une ligne rouge en traits mixtes, etc.).

Pour obtenir le quadrillage, on utilise grid on.

On peut aussi récrire la fonction précédente :

```
>> fun=inline('1/(1+x^2)','x');
```

ou

```
>> fun=@(x)[1/(1+x^2)];
```

Ou

```
function y=fun(x)
y=1/(1+x^2);
end
```

Définition	Évaluation	Tracé
fun='1/(1+xˆ2)'	y=eval(fun)	fplot(fun,[-2,2]) fplot('fun',[-2,2])
fun=inline('1/(1+xˆ2)')	y=fun(x) y=feval(fun,x) y=feval('fun',x)	fplot(fun,[-2,2]) fplot('fun',[-2,2])
fun=@(x)[1/(1+xˆ2)]	y=fun(x) y=feval(fun,x) y=feval('fun',x)	fplot(fun,[-2,2]) fplot('fun',[-2,2])
function y=fun(x) y=fun(x) y=1/(1+xˆ2); end	y=feval(@fun,x) y=feval('fun',x)	fplot('fun',[-2,2]) fplot(@fun,[-2,2])

2) Les zéros

Pour calculer un zéro d'une fonction fun au voisinage d'une valeur donnée x0, réelle ou complexe.

fzero(fun,[x0 x1]),

```
>> fun=@(x)[x^2 - 1 + exp(x)];
>> fzero(fun,-1)
ans =
-0.7146
>> fzero(fun,1)
ans =
5.4422e-18
```

zeros(n), création d'une matrice de zéros

3) La fonction find

Dans sa version la plus simple : retourne un vecteur colonne d'indices des éléments qui satisfont une condition logique.

```
>> A = magic(4) % carré magique
A = 16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
>> i = find(A > 8);
i = 1
3
6
8
10
12
13
15
>> A(i) = 100
A = 100 2 3 100
5 100 100 8
100 7 6 100
```

4 100 100 1
>> **A(find(A > 8)) = 100; % version courte**
>> **A(A > 8) = 100; % encore plus courte**

Version **lignes x colonnes :** retourne **deux vecteur colonnes d'indices** (les indices lignes et colonnes) des éléments qui satisfont une condition logique

Particulièrement utile avec les **matrices creuses**

>> **A = magic(4) % carré magique**
A = 16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
>> **[li, col] = find(A > 12)**
li = 1
4
4
1
col = 1
2
3
4
>> **A(li,col) = 200**
A = 200 200 200 200
5 11 10 8
9 7 6 12
200 200 200 200

4) Les polynômes

Les polynômes sont des fonctions très particulières auxquellesMATLAB dédie la toolbox polyfun. La commande polyval, permet d'évaluer un polyval polynôme en un ou plusieurs points. Ses arguments en entrée sont un vecteur p et un vecteur x, où les composantes de p sont les coefficients du polynôme rangés en ordre des degrés décroissants, de a_n à $a0$, et les composantes de x sont les points où le polynôme est évalué. Le résultat peut être stocké dans un vecteur y en écrivant :

>> y = polyval(p,x)

5) Autres fonctions

Plein d'autres fonctions existes sur Matlab. Voir annexe de ce document avec la liste des fonctions.

Chapitre3 : Script et fonctions

1) Les scripts MATLAB

MATLAB est un interpréteur, c'est-à-dire qu'une fois le logiciel démarré, l'écran présente une invite ' »' à l'utilisateur qui peut entrer une commande que MATLAB « interprète » et exécute dès sa terminaison signalée par l'appui sur la touche <entrée>.

Cependant il est souvent intéressant de conserver des suites de commandes pour pourvoir les faire exécuter, simplement en tapant un nom qui leur sera associé.

Le mécanisme proposé par MATLAB autorisant ce procédé est l'écriture à l'aide d'un éditeur ASCII de la suite des commandes et de la sauver sous un nom de fichier d'extension .m.

2) Les fonctions MATLAB

On appelle fonction une suite d'énoncés portant un nom et pouvant présenter une ou plusieurs valeurs résultantes.

La définition du nom est ma « déclaration de fonction » et doit être placée, comme les scripts, dans des fichiers séparés d'extension .m. L'utilisation de la fonction dans le cours du programme est « l'appel de fonction ».

Ex :

```
function [x1,x2, nb] = equ2(a,b,c)


if nargin ~=3
        error('equ2 demandes 3 arguments ;
else
        x1 = [] ;
        x2 = [] ;
        if a == 0
        [x1,nb] = equ1(b,c) ;
        else
                delta = b.*b-4.*a.*c;
                if delta >=0
                        delta = sqrt(delta);
                        x1 = (-b-delta)./(2.*a);
                        x2 = (-b+delta)./(2.*a);
                        nb =2;
                else
```

```
            nb =0;
           end;
    end;
    end
```

3) **Paramètres, objets locaux et globaux**

3.1) Paramètres

Un paramètre d'entrée d'une fonction est semblable à une variable locale de cette fonction ; si le paramètre effectif est une variable du programme, la valeur de cette variable à la sortie de la fonction sera inchangée par rapport à celle qu'elle possédait auparavant et ceci quel que soit le traitement subi par le paramètre dans le corps de la fonction. De plus le paramètre effectif peut être une constante ou une expression.

Un paramètre de sortie d'une fonction est semblable à une variable utilisée au niveau appelant cette fonction, le paramètre effectif est obligatoirement une variable du programme ou un paramètre connu dans le bloc appelant, et toute modification du paramètre formel dans le corps de la fonction sera répercutée à la sortie sur le paramètre réel.

Une fonction MATLAB peut utiliser un nombre variable de paramètres d'entrée et de sortie.

Les deux fonctions nargin et nargout permettent de savoir le nombre des paramètres passés à une fonction.

3.2) Objets globaux

Un objet global en MATLAB est un objet, qui sans être un paramètre de quelque fonction que ce soit, est partagé par un ensemble de fonctions et/ou de scripts.

Un tel objet doit être déclaré à l'aide d'une directive global dans chaque fonction ou script où l'on désire l'utiliser. Il est inconnu partout ailleurs.

3.3) Objets locaux

Un objet local est un objet qui est défini dans une fonction, il est connu partout à l'intérieur de cette fonction à partir de sa première occurrence, l'espace mémoire qu'il occupe est libéré dès l'achèvement du code de la fonction.

Chapitre 4 : Les tableaux et les boucles itératives

1) Déclarer et utiliser des matrices

Le constructeur

Voici quelques écritures définissant vecteurs ou matrices :

>> x = [2 4 5 8 12] ; vecteur ligne à 5 éléments

>> a = [1 2 3 ; 4 5 6 ; 7 8 9] ; matrice 3x3

>> y = [x x] ; vecteur ligne à 10 éléments

>> z = [x ; x] ; matrice 2x5

La virgule ou le blanc séparant les éléments d'une même ligne, le point-virgule ou le retour chariot séparant les lignes.

L'opérateur

>> x = 1 :5 ; vecteur ligne à 5 éléments

x = [1 2 3 4 5]

>> z = 0 :0.0 :0.5 vecteur à 6 éléments

Z = [0.0 0.1 0.2 0.3 0.4 0.5] ;

>> a = [1 :3 ; 4 :6 ; 7 :9] ; matrice 3x3

a = [1 2 3 ; 4 5 6 ; 7 8 9] ;

Matrice spéciales

Certaines fonctions prédéfinies permettent de construire des matrices spéciales souvent utilisées :

ones	Construit des matrices remplies de 1.
eye	Construit une matrice identité
zeros	Construit une matrice nulle
rand	Construit une matrice dont les éléments suivent une loi uniforme sur [0,1]
randn	Construit une matrice dont les éléments suivent une loi normale réduite
diag	Construit une matrice diagonale ou extrait une diagonale
triu	Extrait la partie triangulaire supérieure
tril	Extrait la partie triangulaire inférieure

Pour chacun d'entre elles 3 syntaxes peuvent être utilisées :

A =zeros(n) matrice n x n

A = zeros(n,m) matrice n x m

A = zeros(size(b)) matrice de même taille que b

Remplissages réguliers

Les fonctions linspace, logspace et meshgrid sont également utiles pour créer des vecteurs ou matrices réalisant des remplissages discrets.

linspace	Points espacés régulièrement
logspace	Points espacés logarithmiquement
meshgrid	Points en grille

linspace, logspace demandent trois paramètres : deb, fin et nb et créent respectivement des vecteurs à nb éléments également espacés entre deb et fin.

meshgrid est une aide efficace au tracé de surfaces : étant donné deux vecteurs de même taille x et y, meshgrid en crée deux autres xx et yy qui sont tels que les coupés (xx(i), yy(i)) décrivent tous les couples (x(i), y(j)).

Taille des matrices

size	Dimension
length	Dimension maximale

Size est une fonction qui permet de connaître la taille courante d'une matrice. En écrivant [n,m] = size(a), on obtient le nombre de lignes et de colonnes de a. length est une fonction analogue qui renvoie le maximum des dimensions.

2) Indexer les matrices

Il y a deux types d'indexation :

A(v1,v2) ou v(v3)

3) La boucle for

La boucle for permet des itérations sur les colonnes de matrices, sa syntaxe est :

E = <exp>

[m,n] = size(e);

For j = 1:n

V = e(:,j);

<ins1>; <ins2>; … <insn>;

End

4) La boucle while

```
n = 10;
f = n;
while n > 1
    n = n-1;
    f = f*n;
end
disp(['n! = ' num2str(f)])
```

Chapitre 5 : Les opérateurs

1) Les opérateurs arithmétiques

’	Transposée de la conjuguée
.’	Transposée
+	Plus binaire ou unaire
-	Moins binaire ou unaire
*	Multiplication de matrices
.*	Multiplication terme à terme
^	Puissance matricielle
.^	Puissance terme à terme
\	Backslash : division matricielle gauche
/	Slash : division matricielle droite
.\	Division gauche terme à terme
./	Division droite terme à terme

2) Opérateurs rationnels et logiques

==	Égal
~=	Différent
<	Moins grand que
>	Plus grand que
<=	Moins grand que ou égal à
>=	Plus grand que ou égal à
&	Et logique
|	Ou logique
~	Négation logique
xor	Ou exclusif logique

3) États de variables numériques

finite	Valeur finie ?
isreal	Partie imaginaire nulle ?
isnan	NaN ?
isinf	Infini ?
ischar	Chaine ?
exist	Y-en-a-t-il un de ce nom ?

- Finite est vrai pour des valeurs finies.

Finite(x) renvoie des 1 partout où x est fini et 0 ailleurs.

- Isreal est vrai pour les valeurs réelles

Isreal(x) renvoie des 1 partout où x est de partie imaginaire nulle, même si x est de type chaîne (char).

- Isnan est vrai pour des non-nombres

Isnan(x) renvoie des 1 partout où x vaut NaN.

- Isinf vrai pour des infinis

Isinf(x) renvoie des 1 partout où x vaut inf ou –inf.

- Ischar est vrai pour les chaînes de caractères.

Il existe d'autres fonctions traitant des chaînes de caractères.

- Exist

Exist('<nom>') renvoie :

0 si le nom n'existe pas

1 si le nom est une variable de l'espace de travail

2 si le nom est un m-fichier sur un des chemins de recherche ou si c'est exactement le nom d'un fichier

3 si le nom est un mex-fichier sur un des chemins de recherche

4 si le nom est une fonction compilée SIMULINK

5 si le nom est une fonction interne de MATLAB

Chapitre 6 : Structures de données

Tableau de structure :

```
a(2).name = 'Renaud'
a =
1x2 struct array with fields:
name
Numbers
>> a(2).numbers % Remplissage par défaut à []
ans =
[]
>> a(2).numbers = rand(3,3); a %tableau ligne !!!!
a =
1x2 struct array with fields:
name
numbers
```

Création par la fonction struct() struct('champ1',val1,'champ2',val2,...) :

a(1,1) ={struct('nom','zorro','age',72)}

ou en utilisant { }

Chapitre 7 : Les chaînes de caractères

1) Fonctions

abs, double	Conversion chaîne → code ascii
char, setstr	Conversion code ascii → chaîne
isstr	Test de type
blanks	Génération de chaînes de blancs
str2mat, strvcat	Création d'une matrice de chaînes

2) Comparaison de chaîne

strcmp	Comparaison
strcmpi	Comparaison sans casse
strncmpi	Comparaison nombre fixé d'éléments
strmatch	Trouve les chaînes correspondantes
findstr	Recherche d'une sous-chaîne
upper	Conversion en majuscule
lower	Conversion en minuscule
isletter	Vrai si caractère alphabétique accentué ou non
isspace	Vrai pour les caractères « blancs »
strrep	Remplacement de chaînes
strtok	Recherche le premier élément délimité

3) Conversions chaînes <--> Nombres

num2str	Conversion nombre (réel/complexe)→ chaîne
int2str	Conversion entier → chaîne
str2num	Interprétation d'une (matrice de) chaînes de nombres comme matrices de nombres
sprintf	Conversion nombre → chaîne avec format
sscanf	Conversion chaîne → nombre de format

4) Évaluation des chaînes et gestion d'erreur

error	Arrêt d'un programme après erreur d'exécution
warning	Avertissement de possibilité d'erreur
eval	Exécution d'une chaîne
feval	Exécution d'une fonction donnée par son nom
lasterr	Dernier message d'erreur

Chapitre 8 : L'algèbre linéaire

1) Analyse des matrices

det	Déterminant d'une matrice carrée
cond	Conditionnement d'une matrice
null	Noyau d'une application linéaire
rank	Rang d'une matrice
orth	Image d'une application linéaire
trace	Trace d'une matrice
norm	Diverses normes matricielles

2) Système linéaire

chol	Décomposition de Cholevsky
lu	Décomposition PLU
inv	Calcul de la matrice inverse
qr	Décomposition QR, produit d'une matrice unitaire et d'une matrice triangulaire supérieure
lsqnonneg	Moindres carrés positifs
lscov	Mondres carrés avec covariance connue

3) Valeurs propres ou singulières

eig	Valeurs et vecteurs propres (généralisées)
polyeig	Valeurs propres généralisées polynômiales
poly	Polynôme caractéristique
svd	Décomposition par valeur singulière
hess	Forme de Hessenberg
schur	Forme Schur
rsf2csf	Bloc diagonale → complexe diagonale
cdf2rdf	Complexe diagonale → bloc diagonale
balance	Rééquilibrage

4) Fonctions matricielles

expm	Exponentielle de matrice
logm	Logarithme naturel d'une matrice
sqrtm	Racine carrée
funm	Évaluation d'une fonction ordinaire comme fonction matricielle

Chapitre 9 : Polynômes, interpolation, et statistiques de base

1) Racines

roots	Polynôme →racines
poly	Racines → polynôme

2) Évaluation

polyval	Évaluation
polyvalm	Évaluation matricielle
polyder	Évaluation des dérivées

3) Décomposition en éléments simples

residue	Décomposition en éléments simples

4) Produit et quotient

conv	Produit de deux polynômes
deconv	Quotient de deux polynômes

5) Approximation

polyfit	Approximation par un polynôme

6) Interpolation

interp1	Interpolation polynômiale à une dimension
interpft	Interpolation polynômiale à une dimension par FFT

7) Interpolation bidimensionnelle

interp2	Interpolation polynômiale à deux dimensions
interp3	Interpolation biharmonique
griddata	Mise en grille de données

8) Opérations statistiques de base

max	Maximum. Appliqué à un vecteur, max renvoie la valeur et optionnellement l'indice du grand élément.
min	Minimum. Comme le précèdent, en remplaçant plus grand par plus petit
mean	Moyenne d'un vecteur
median	Médiane d'un vecteur. La médiane m est l'élément pour lequel le vecteur possède autant d'éléments qui lui sont plus petits que d'éléments qui lui sont plus grand.

std	Écart type d'un vecteur. C'est la racine carrée de la moyenne des carrés des écarts à la moyenne
sort	Tri par ordre croissant de valeur. Cette fonction peut fournir optionnellement un tableau d'indices d'éléments dans le vecteur initial.
sum	Somme des éléments
prod	Produit des éléments
cumsum	Sommes cumulées des éléments. Cette fonction calcule un vecteur des sommes partielles du paramètre.
cumprod	Produits cumulés des éléments. Cette fonction calcule un vecteur des produits partiels du paramètre.

corrceof	Coefficients de corrélation C = corrcoef(x) est une matrice formée à partir d'un x tel que chaque ligne est une observation et chaque colonne une variable.
cov	Matrice de covariance
subspace	Angle entre deux sous-espaces.

Chapitre 10 : Entrées et sorties

1) Fonctions élémentaire

input	Entrée clavier évaluée ou non
keyboard	Outil de déverminage
menu	Menu simple
pause	Attente
disp	Sortie écran
load	Entrée fichier
save	Sortie fichier

2) Fonctions de bas niveau

	Fermeture et ouverture
fopen	Ouverture d'un fichier
fclose	Fermeture d'un fichier
	IO non formatées
fwrite	Écriture de données binaires
fread	Lecture de données binaires
	IO formatées
fscanf, textread	Lecture de données formatées
fprintf	Écriture de données formatées
fgetl	Lecture d'une ligne avec rejet du caractère de fin de ligne
fgets	Lecture d'une ligne avec conservation du caractère de fin de ligne
ferror	Demande du statut d'erreur d'I/O
feof	Test de fin de fichier
	Positionnement
fseek	Mise à une position dans un fichier
ftell	Demande de la position courante
frewind	Remise au début de fichier (rembobinage)

Interface graphique

Chapitre 11 : 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

Graphes spécifiques et annotations

polar	Coordonnées polaires
bar	Barres
stem	Traits verticaux surmontés d'un petit cercle en chaque donnée
stairs	Marches d'escalier
errorbar	Barre d'erreur
hist	Histogramme
rose	Histogramme angulaire
feather	Plumes…
comet	Trajectoire de comète le long des données (animation)
fplot	C'est la version système de funplot.

Tracés de fonction 3D

contour	Tracé de contours
contour3	Lignes de niveau 3D
clabel	Cotes d'élévation des lignes de niveau
pcolor	Tracé par pseudo-couleurs (en échiquier)
quiver	Tracé « quiver »

Surfaces et volumes

mesh	Surface définie par une grille 3D
meshc	mesh + contour
meshz	mesh + plan zéro
surf	Surface ombrée
surfc	surf + contour
surfl	Surface ombrée avec éclairage
waterfall	Tracé en chute d'eau
Slice	Visualisation volumétrique

Annotations

zlabel, xlabel, ylabel

Organiser les graphes

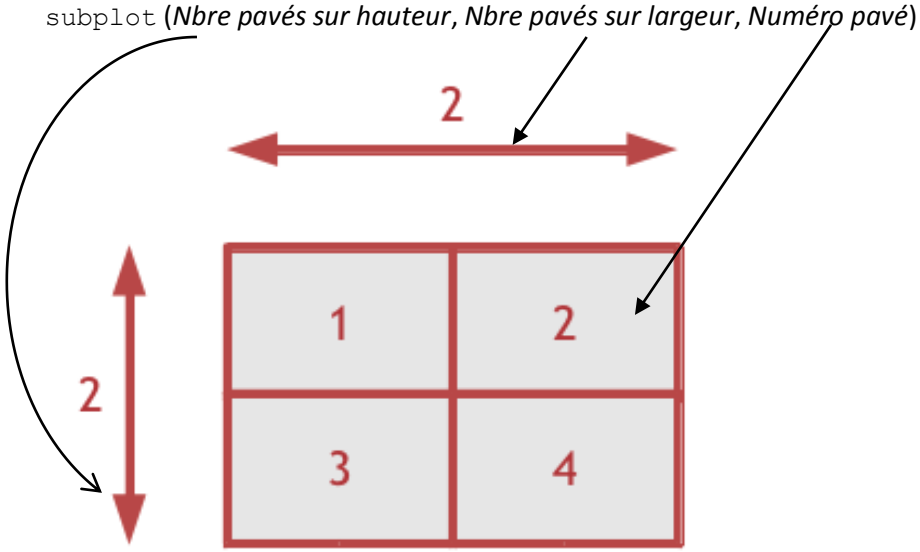
hold	Conservation/effaçage des tracés précédents
subplot	Axes multiples dans une même fenêtre
figure	Nouvelle fenêtre

Dans le même axe, on utilise hold on.

SUBPLOT

La syntaxe est la suivante:

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



Placement de texte

title	Titre du graphe
xlabel	Label de l'axe x
ylabel	Label de l'axe y
zlabel	Label de l'axe z
text	Annotation
gtext	Placement de texte à l'aide de la souris
grid	Grilles

La syntaxe générale est la suivante :

text(x,y,'string')

text(x,y,'string',propriétés)

Il est possible de mettre des caractères Latex :

Character Sequence	Symbol	Character Sequence	Symbol	Character Sequence	Symbol
\alpha	α	\upsilon	υ	\sim	~
\beta	β	\phi	ϕ	\leq	≤
\gamma	γ	\chi	χ	\infty	∞
\delta	δ	\psi	ψ	\clubsuit	♣
\epsilon	ε	\omega	ω	\diamondsuit	♦
\zeta	ζ	\Gamma	Γ	\heartsuit	♥
\eta	η	\Delta	Δ	\spadesuit	♠
\theta	θι	\Theta	Θ	\leftrightarrow	↔
\vartheta	ϑ	\Lambda	Λ	\leftarrow	←
\iota	ι	\Xi	Ξ	\uparrow	↑
\kappa	κ	\Pi	Π	\rightarrow	→
\lambda	λ	\Sigma	Σ	\downarrow	↓
\mu	μ	\Upsilon	Υ	\circ	º
\nu	ν	\Phi	Φ	\pm	±
\xi	ξ	\Psi	Ψ	\geq	≥
\pi	π	\Omega	Ω	\propto	∝
\rho	ρ	\forall	∀	\partial	∂
\sigma	σ	\exists	∃	\bullet	•
\varsigma	ς	\ni	∋	\div	÷
\tau	τ	\cong	≅	\neq	≠
\equiv	≡	\approx	≈	\aleph	ℵ

\Im	ℑ	\Re	ℜ	\wp	℘
\otimes	⊗	\oplus	⊕	\oslash	⊘
\cap	∩	\cup	∪	\supseteq	⊇
\supset	⊃	\subseteq	⊆	\subset	⊂
\int	∫	\in	∈	\o	o
\rfloor	⌋	\lceil	⌈	\nabla	∇
\lfloor	⌊	\cdot	·	\ldots	...
\perp	⊥	\neg	¬	\prime	´
\wedge	∧	\times	x	\0	∅
\rceil	⌉	\surd	√	\mid	|
\vee	∨	\varpi	ϖ	\copyright	©
\langle	⟨	\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`.

Contrôle d'aspect des axes

- **axis([xmin xmax ymin ymax])** permet de fixer les limites des axes 2D
  axis([xmin xmax ymin ymax zmin zmax]) permet de fixer les limites des axes 3D
- **axis auto**

- **axis manual** gèle les axes qui restent constants indépendamment des données qui y sont tracées, si graphe est un mode « conservation »
- **axis tight** règle le rapport d'aspect en sortie que toutes les unités soient identiques dans les différentes directions
- **axis fill** règle les limites sur les intervalles des données
- **axis ij** crée des axes matriciels : i vertical vers le bas, j horizontal. axis xy rétablit la valeur par défaut
- **axis off** cache les lignes, les ticks, et les labels des axes on rétablit leur visibilité

Chapitre 12 : figure

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

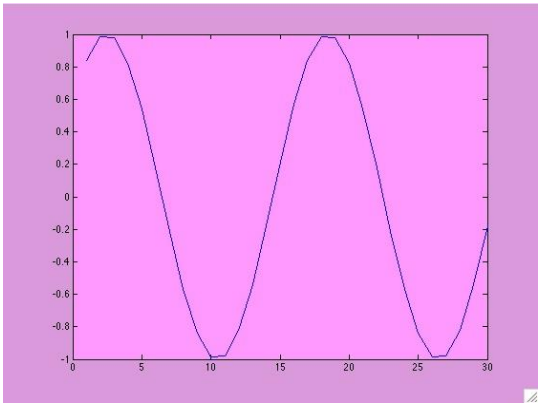
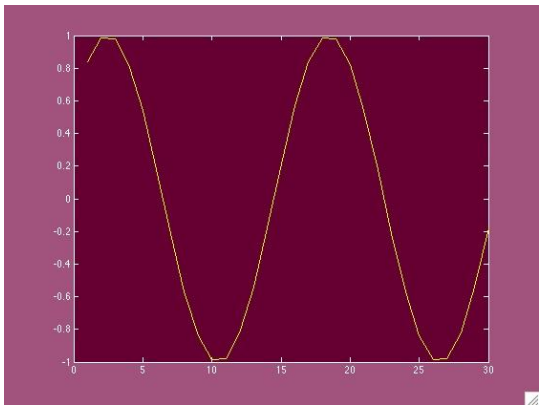
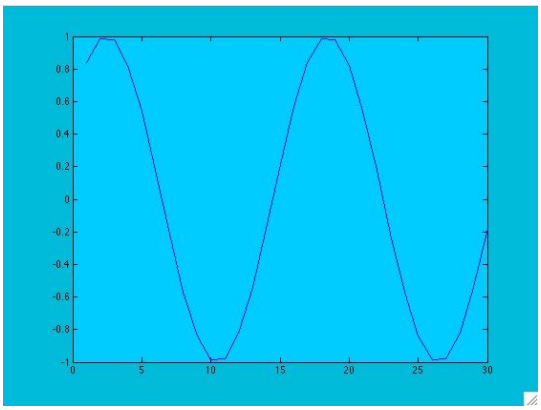
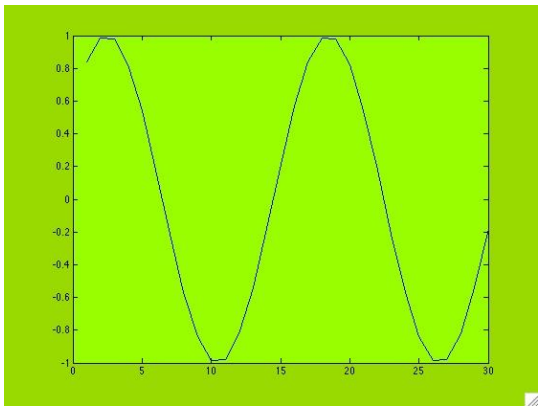
Cette fonction change toute les plots de la figure.

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

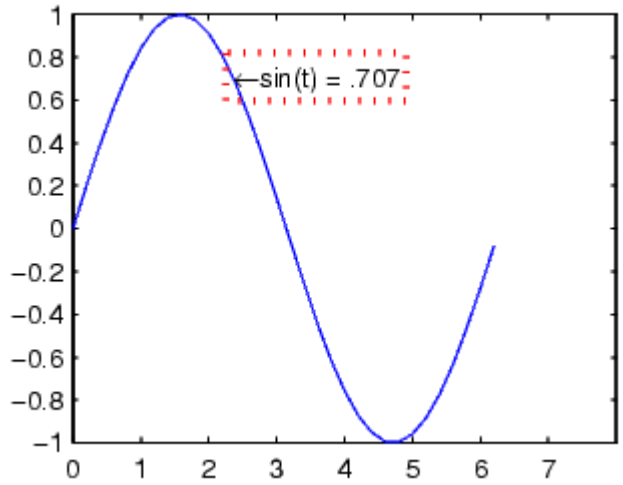
HorizontalAlignment viewed with the VerticalAlignment set to middle (the default).

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

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

Par exemple comme code:

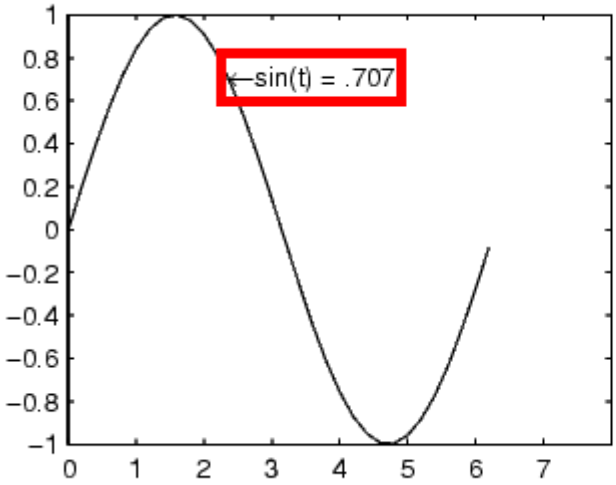
```
text(3*pi/4,sin(3*pi/4),...
     '\leftarrowsin(t) = .707',...
     'EdgeColor','red',...
     'LineWidth',2,...
     'LineStyle',':');
```



- **LineWidth** : scalar(points)

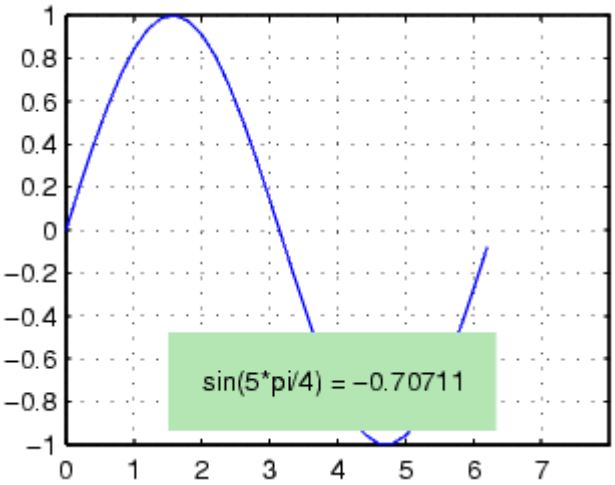
Code exemple:

```
text(3*pi/4,sin(3*pi/4),...
'\leftarrowsin(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).

- **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.

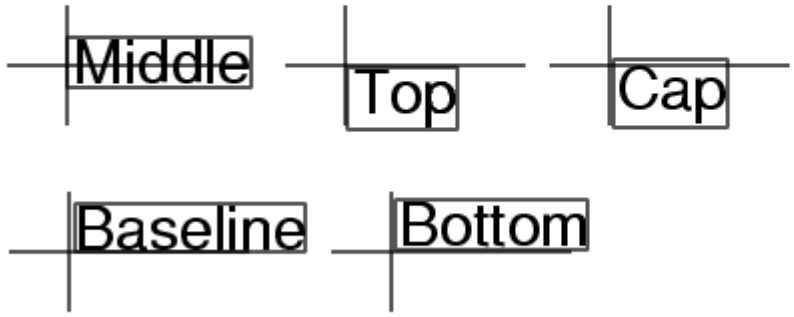
- `Normalized` units map the lower left corner of the rectangle defined by the axes to (0,0) and the upper right corner to (1.0,1.0).
- `pixels`, `inches`, `centimeters`, and `points` are absolute units (1 point = $^1/_{72}$ inch).
- `data` refers to the data units of the parent axes.

- **VericalAlignment** : `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).



Chapitre 13 : les images

- Image I/O

(1) Image standard:

imread : pour lire une image

- A = imread(filename, fmt) ou A = imread(filename) avec filename le chemin du fichier image et fmt est l'extension. On ressort en A la matrice d'intensité des pixels.

Ex : Y = imread('lena.jpg')ou Y = imread('lena.jpg', 'jpg')

- [X,map] = imread(filename,fmt) ou [X,map] = imread(finename) avec filename le chemin du fichier image et fmt est l'extension. On ressort X : la matrice d'intensité des pixels et map : la colorimétrie.

Ex :

```
[X, map] = imread('peppers.png');
image(x) % pour l'affichage
colormap(map) % gestion de la couleur de la LUT
```

- Les formats pris en compte :

'jpg', 'jpeg'	« Joint Photographic Experts Group » (JPEG)
'tif', 'tiff'	« Tagged Image File Format » (TIFF)
'bmp'	Bitmap de Windows (BMP)
'png'	« Portable Network Graphics »
'hdf'	« Hierarchical Data Format » (HDF)
'pcx'	« Windows Painbush » (PCX)
'xwd'	Dump de X Window (XWD)

imwrite : pour créer une image

- imwrite(A,filename,fmt) avec A : la matrice de pixel, filename : le nom de l'image qui va être crée, fmt : le nom de l'extension

iminfo : donne les informations sur l'image, par exemple sa taille, la date de création…

- info = imfinfo(filename,fmt) avec filename : du fichier entrée de l'image, et fmt : est l'extension
- info = imfinfo(filename) avec filename : du fichier entrée de l'image

(2) Image DICOM :

dicomread : permet de récupérer la matrice d'intensité de pixel (le niveau de gris)

- X = dicomread(filename) avec filename : le chemin et le nom du fichier DICOM
- X = dicomread(filename, 'frames', v) avec filename : le chemin et le nom du fichier DICOM, frames : signifie une image, v : nombre d'image dans le fichier DICOM à 90%, il y a qu'une seule image (modalité MR, CR) et par exemple des images compressés comme US (ultrasons)

Exemple :

X = dicomread('C:/dicom.dcm', 'frames',1);
dicominfo : affiche les métadonnées de DICOM

- info = dicominfo(filename) avec filename : le chemin et le nom du fichier DICOM

Exemple :
info = dicominfo('CT-MONO2-16-ankle.dcm')

- Image display
  - **image**: affiche l'image

Ex:
```
figure
ax(1) = subplot(1,2,1);
rgb = imread('ngc6543a.jpg');
image(rgb); title('RGB image')
ax(2) = subplot(122);
im = mean(rgb,3);
image(im); title('Intensity Heat Map')
colormap(hot(256))
linkaxes(ax,'xy')
axis(ax,'image')
```

- **imagesc**: mise à l'échelle et affichage de l'image

Ex :

```
load clown
imagesc(X)
colormap(gray)
```

- **imshow**: affichage de l'image

Ex:

```
I = imread('cameraman.tif');
imshow(I,[])
```

- **subimage**: Display multiple images in a single figure even if they have different colormaps

```
load trees
[X2,map2] = imread('forest.tif');
subplot(1,2,1), subimage(X,map)
subplot(1,2,2), subimage(X2,map2)
```

- Outils

  - **Imtool**: Fournit la matrice de pixel, région de pixel, distance, information de pixel, permet d'ajuster le contraste, afficher l'image.

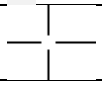
Chapitre 14 : Changer le pointer de la souris une figure

Exemple de code :

```
hf = figure('position',[100 100 400 200], 'pointer','crosshair', 'color',[1 0 0], ...
                'Name','Mon premier interface')
```

L'attribut 'pointer' peuvent être utiliser soient dans une figure ou dans un set comme set(gcf, 'Pointer', 'arrow')

La liste des pointeurs possibles est la suivante:

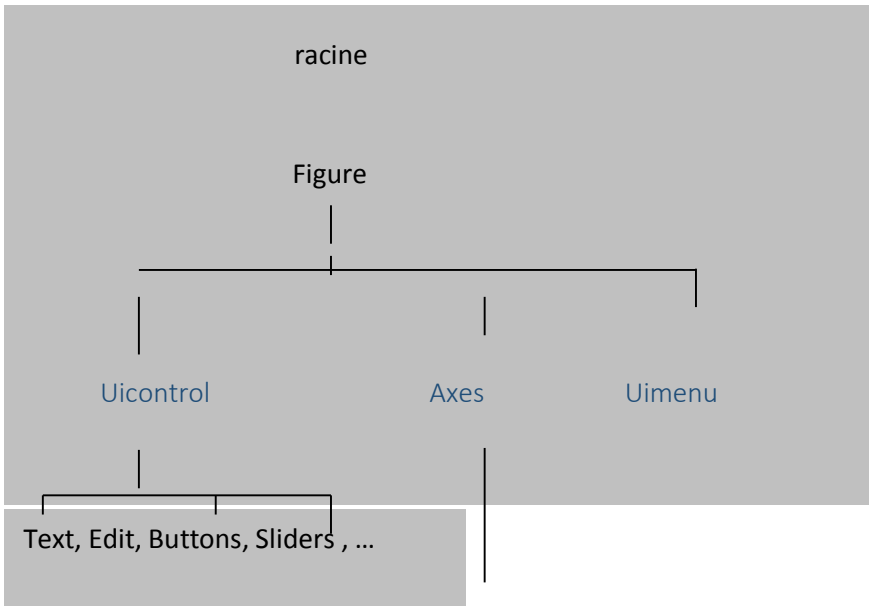
Spécification	Symbole
ibeam	
crosshair	
arrow	
watch	
topl	
topr	
botl	
botr	
circle	
cross	
fleur	
left	
right	
top	
bottom	
fullcross	
custom	

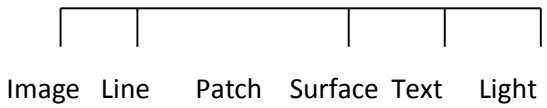
IHM

Chapitre 15 : les bases

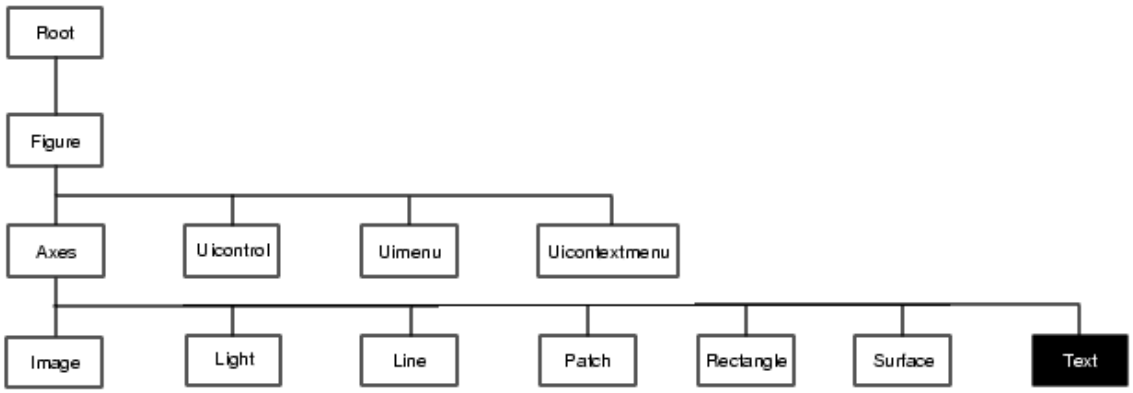
I) Structure d'arbre:

Un GUI se présente comme une structure arborescente (ci-dessous) composée d'objets d'interface:





Autre représentation :



II) Objets interface :

On utilise les objets suivants dans la suite, ce n'est pas une liste exhaustive:

figure, text, axes, line, edit, slider, button, checkbox...

III) Propriétés :

Chaque **objet** possède un ensemble de propriétés généralement programmable qui fixent l'apparence graphique et les réactions de l'objet aux sollicitations de l'utilisateur.

Les propriétés peuvent être des chaînes de caractères, des vecteurs de valeurs numériques, spécifiés selon le format courant:

Par exemple, **Xtick = [0 : 0.2:1]** ou **Xgrid = 'on'**. Si chaque type d'objet possède des propriétés propres, certaines propriétés sont communes à tous les objets : un objet **text** a un nom (propriété **Tag**), une chaîne de caractères qu'il affiche (propriété **String**), police de caractères **FontName** de taille **Fontsize**... Pour retrouver la valeur d'une propriété, il faudra en spécifier le nom dans une chaîne de caractères. L'éditeur d'interface **guide** ne teste pas les majuscules et les trois premières lettres suffisent pour retrouver une propriété : on pourra taper **'str'** pour la propriété **'String'** par exemple.

IV) Le callbacks : Réactivité de l'interface

Parmi les propriétés des objets de l'interface, les Callbacks contiennent des scripts ou des fonctions MATLAB pour programmer les réactions de l'interface aux commandes de l'utilisateur.

Ainsi, imaginons un **PushButton** nommé `'Bouton'` auquel on a associe le Callback : `grid on`. Cliquer sur **Bouton** provoque le tracé d'une grille sur les axes de tracé courants. Le Callback `close(gcf)` fermerait l'interface graphique, `cla` effacerait les tracés, etc …

V) Les Handers : Identificateurs des objets

Les objets étant créés lors de la constitution de l'interface, ou dynamiquement durant l'exécution, on leur associe lors de la création un identificateur unique, qu'on appelle le `Handler` et qui permet de les manipuler. Certains handlers sont réservés et mis à jour en permanence :

gcf : **attaché à la figure courante**

gca : **axes de tracé courants**

gcbf : **figure activée (dans laquelle on clique)**

Pour retrouver dans un script le handler d'un objet de l'interface dont on connaît une propriété qui le caractérise, on peut utiliser la fonction `findobj` **:**

h = findobj(gcf, 'Tag', 'Fig1') par exemple, h handleer de Fig1.

Chapitre 16 : Approfondissement

1) Différentes termes à connaître :

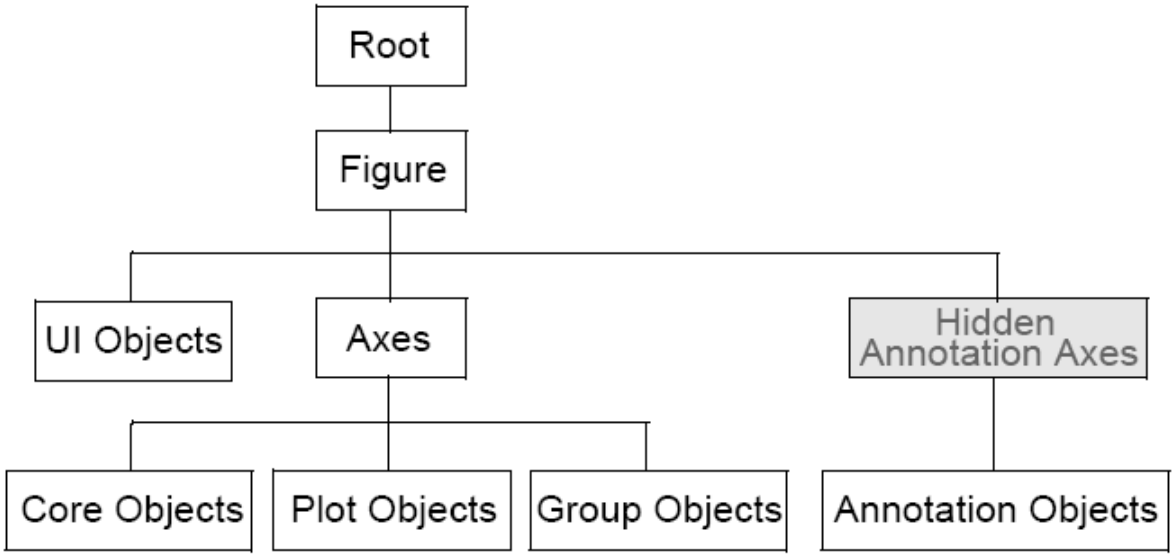


Figure 1 : Arborescence de la gestion d'affichage

Descriptions :

- Root :

Au sommet de la hiérarchie se trouve l'objet Root. Cet objet est invisible (on peut se le représenter comme étant la surface de l'écran de l'ordinateur). L'utilisateur n'interagit que très rarement avec cet objet.

- Objets Figure :

Les objets Figure sont les conteneurs visibles où sont disposés tous les autres objets enfants. Ces objets sont couramment appelés "fenêtres". Plusieurs objets Figure peuvent être ouverts simultanément et peuvent éventuellement communiquer entre eux.

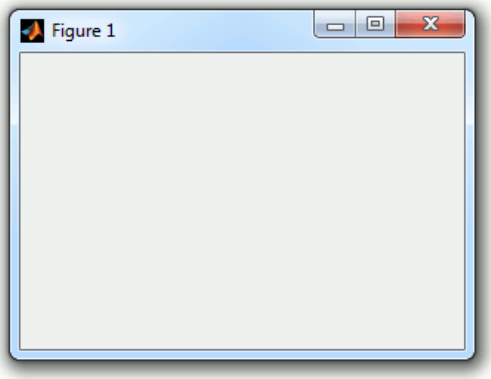


Figure 2 : Création figure

- Objets Axes :

  Les objets Axes sont les zones de traçage des graphiques (2D ou 3D). Un objet Figure peut contenir plusieurs objets Axes simultanément.

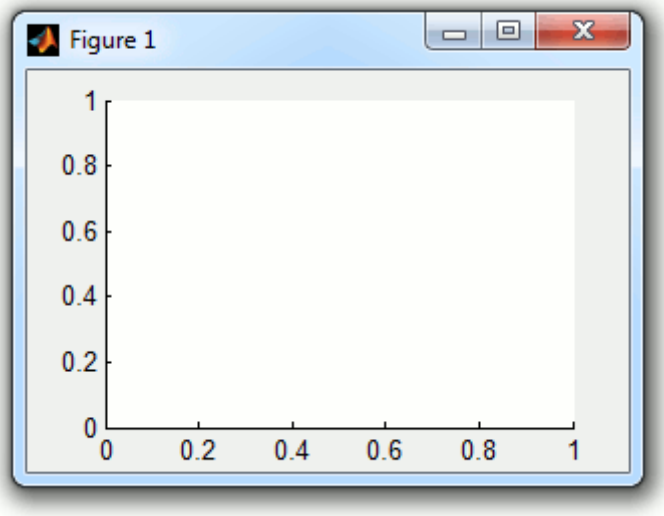


Figure 3 : Création du axes sur figure

- Objets UI :

Au même niveau hiérarchique que les objets Axes, on trouve les objets UI (pour User Interface). Certains de ces objets (comme les boutons, les menus, les cases à cocher) permettent à l'utilisateur d'interagir avec l'interface graphique grâce à la souris ou au clavier. D'autres objets (comme les panels, les tables...) servent à la mise en forme de l'interface graphique.

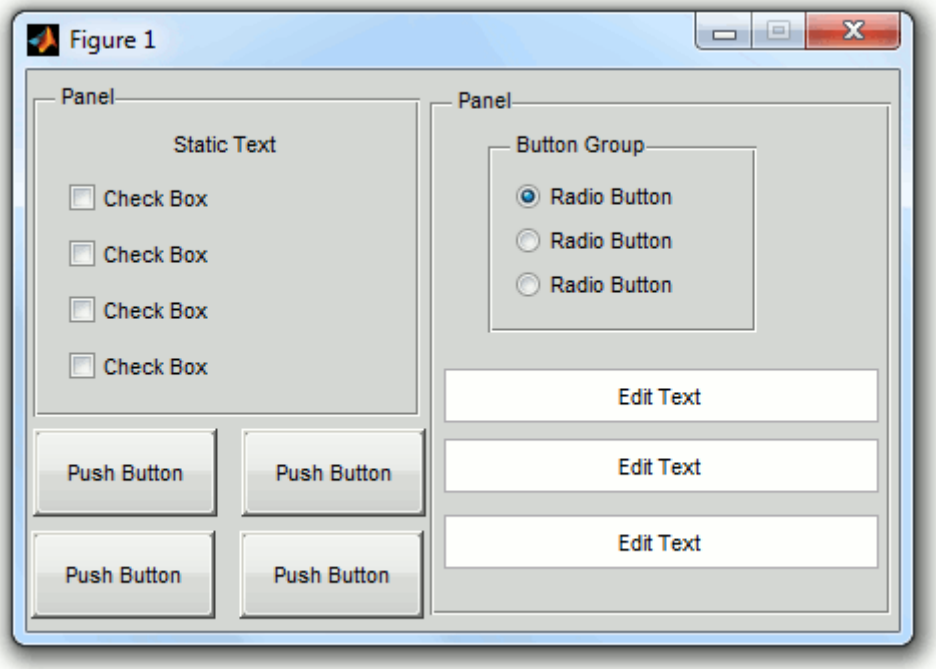


Figure 4 : Les différents uicontrol existant

2) Quelques fonctions utiles :

a) Attributs :

- 0 - root object handle (the screen)
- gcf – returns the handle of the current figure
- gca - returns the handle of the current axis in the current figure
- gco - returns the handle of the current object in the current figure
- gcbo - returns the handle of the object whose callback is currently executing
- gcbf - returns the handle of the figure that contains the object whose callback is currently executing
- findobj(handles,'PropertyName',PropertyValue) – return objects with specific properties

- 0- racine
- **gca** : récupère l'identifiant de l'objet Axes courant
- **gcbf** : récupère l'identifiant de l'objet Figure où se trouve l'objet graphique dont l'action est en cours
- **gcbo** : récupère l'identifiant de l'objet graphique dont l'action est en cours
- **gcf** : récupère l'identifiant de l'objet Figure courant
- **gco** : récupère l'identifiant de l'objet graphique courant
- **findobj(handles,'PropertyName',PropertyValue)** : récupère objet avec les propriétés spécifiques

b) Les get et set :

- **Return a list of all object properties and their current values:**
  –get(handle)
- **Return current value of an object property:**
  –get(handle, 'PropertyName')
  –Exemple: get(gcf, 'Color')
- **Return a list of all user-settable object properties and their current values:**
  –set(handle)

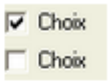
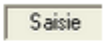
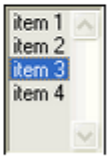
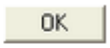
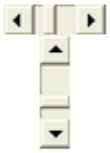
•**Return a list of all possible values for an object property:**
  –set(handle,'PropertyName')
  –Exemple: set(gca, 'XDir')

•**Set an object property to a new value:**
  –set(handle, 'PropertyName', 'NewPropertyValue')
  –Exemple: set(gca, 'XDir', 'Reverse')

c) Uicontrol:
Exemple:

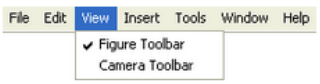
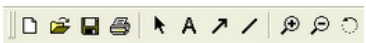
```
uicontrol('Style', 'popup',...
          'String', 'jet|hsv|hot|cool|gray',...
          'Position', [20 340 100 50],...
          'Callback', @setmap);   % Popup function handle
callback
                        % Implemented as a local function
```

Voici un tableau qui résume les différents constructeurs.

Nom	Désignation MATLAB	Aperçu
Case à cocher	checkbox	
Zone de texte éditable	edit	
Cadre	frame	
Liste	listbox	
Menu déroulant	popupmenu	
Bouton	pushbutton	
Bouton Radio	radiobutton	
Barre de défilement	slider	
Zone de texte non éditable	text	
Bouton à 2 états	togglebutton	

d) Quelques objets utiles:

Les autres objets regroupent les menus, les barres d'outils... Ils sont créés à l'aide des fonctions UIMENU, UICONTEXTMENU, UIPUSHTOOL, UITOGGLETOOL, UITOOLBAR.

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 17 : Poignées graphiques

1) Les objets graphiques et leurs propriétés

Les axes : axes
Les contrôles : uicontrol
L'éclairage : light
Les figures : figure
Les images : image
Les lignes : line
Les menus : uimenu
Les menus contextuels : uicontextmenu
Les patchs : patch
La racine : root
Les rectangles : rectangle
Les surfaces : surface
Les textes : text

Chapitre 18 : 1er exemple à la main

On construit chaque composante graphique.
Pour un débutant, ce n'est pas toujours facile à utiliser.

Quelques méthodes utiles :

- L'utilisation des variables globales : bien que ce type de variable soit excessivement simple à utiliser, on tend toujours à proscrire leur utilisation. Leur totale visibilité constitue un risque majeur d'erreur de programmation si l'on ne prend pas bien soin d'assurer l'unicité de leur nom.
- L'utilisation des fonctions **setappdata**, **getappdata** et **findobj** : cette méthode est très flexible et très fiable si le choix du nom des variables d'application est fait judicieusement. Cette méthode peut aussi être utilisée dans le cadre d'interfaces complexes à plusieurs fenêtres. Dans ce cas, il est commode de stocker les variables d'application dans l'objet graphique Root (0) pour leur assurer une visibilité totale (il faudra veiller à choisir des noms explicites dans ce cas).
- L'utilisation des fonctions **guidata/guihandles** : cette méthode est également très flexible et peut être utilisée avec du code généré par le GUIDE.
- L'utilisation des fonctions imbriquées (nested functions) : cette méthode est parfaitement adaptée si le projet ne comporte qu'un seul fichier.

Les exemples suivants sont pour faire un compteur :

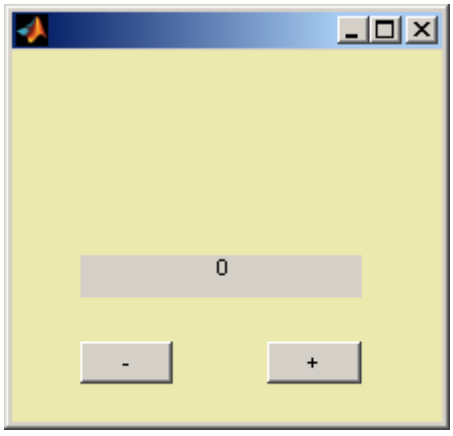


Figure 5: exemple de compteur

1) Variables globales

Le fichier GUI_VAR_GLOBALES.M montre l'utilisation des variables globales pour gérer les variables et les identifiants des objets graphiques pendant l'exécution de l'interface graphique. Une fois les variables définies, en utilisant le mot clé global, dans toutes les fonctions et sous-fonctions, leur manipulation est très aisée. Elles sont constamment visibles dans toutes les

fonctions. Il n'y a donc pas de gestion à proprement dit. Il suffit de les utiliser simplement sans se soucier de leur accessibilité.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%DEBUT DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gui_var_globales

% Définition de nCompteur et handles comme variables globales dans chaque
fonction et sous-fonction
% nCompteur : valeur courante du compteur (scalaire)
% handles : identifiants des objets graphiques (vecteur)
global nCompteur handles

% Initialisation de la variable représentant la valeur courante du compteur
nCompteur à 0
nCompteur=0;

% Création de l'objet Figure
handles(1)=figure('units','pixels',...
    'position',[250 250 500 500],...
    'color',[0.925 0.913 0.687],...
    'numbertitle','off',...
    'name','[GUI] Utilisation des variables globales',...
    'menubar','none',...
    'tag','interface');

% Création de l'objet Uicontrol Pushbutton -
handles(2)=uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.1 0.1 0.1 0.05],...
    'string','-',...
    'callback',@retrancher,...
    'tag','bouton-');

% Création de l'objet Uicontrol Pushbutton +
handles(3)=uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.3 0.1 0.1 0.05],...
    'string','+',...
    'callback',@ajouter,...
    'tag','bouton+');

% Création de l'objet Uicontrol Text résultat
handles(4)=uicontrol('style','text',...
    'units','normalized',...
    'position',[0.1 0.2 0.3 0.05],...
    'string','0',...
    'tag','resultat');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function retrancher(obj,event)
```

```
% Définition de nCompteur et handles comme variables globales dans chaque
fonction et sous-fonction
% nCompteur : valeur courante du compteur (scalaire)
% handles : identifiants des objets graphiques (vecteur)
global nCompteur handles

% Diminution de la valeur de nCompteur
nCompteur=nCompteur-1;

% Actualisation de la propriété String de l'objet Uicontrol Text résultat
set(handles(4),'string',num2str(nCompteur));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION AJOUTER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ajouter(obj,event)

% Définition de nCompteur et handles comme variables globales dans chaque
fonction et sous-fonction
% nCompteur : valeur courante du compteur (scalaire)
% handles : identifiants des objets graphiques (vecteur)
global nCompteur handles

% Augmentation de la valeur de nCompteur
nCompteur=nCompteur+1;

% Actualisation de la propriété String de l'objet Uicontrol Text résultat
set(handles(4),'string',num2str(nCompteur));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION AJOUTER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

2) setappdata, getappdata et findobj:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gui_appdata_findobj

% Création de l'objet Figure
figure('units','pixels',...
    'position',[250 250 500 500],...
    'color',[0.925 0.913 0.687],...
    'numbertitle','off',...
    'name','[GUI] Utilisation de SETAPPDATA / GETAPPDATA',...
    'menubar','none',...
    'tag','interface');

% Création de l'objet Uicontrol Pushbutton -
uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.1 0.1 0.1 0.05],...
    'string','-',...
```

```
    'callback',@retrancher,...
    'tag','bouton-');

% Création de l'objet Uicontrol Pushbutton +
uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.3 0.1 0.1 0.05],...
    'string','+',...
    'callback',@ajouter,...
    'tag','bouton+');

% Création de l'objet Uicontrol Text résultat
uicontrol('style','text',...
    'units','normalized',...
    'position',[0.1 0.2 0.3 0.05],...
    'string','0',...
    'tag','resultat');

% Initialisation de la valeur représentant la valeur courante du compteur
nCompteur à 0
nCompteur=0;

% Enregistrement direct de nCompteur dans les données d'application de
l'objet Figure
setappdata(gcf,'valeur_de_nCompteur',nCompteur);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function retrancher(obj,event)

% Récupération directe de nCompteur depuis les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
nCompteur=getappdata(gcbf,'valeur_de_nCompteur');

% Diminution de la valeur de nCompteur
nCompteur=nCompteur-1;

% Récupération de l'identifiant de l'objet Uicontrol Text résultat enfant
de l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
h=findobj('parent',gcbf,'style','text','tag','resultat');
% Modification de sa propriété String
set(h,'string',num2str(nCompteur));

% Enregistrement de la nouvelle valeur de nCompteur dans les données
d'application de l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
setappdata(gcbf,'valeur_de_nCompteur',nCompteur);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION AJOUTER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function ajouter(obj,event)

% Récupération directe de nCompteur depuis les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
nCompteur=getappdata(gcbf,'valeur_de_nCompteur');

nCompteur=nCompteur+1;

% Récupération de l'identifiant de l'objet Uicontrol Text résultat enfant
de l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
h=findobj('parent',gcbf,'style','text','tag','resultat');
% Modification de sa propriété String
set(h,'string',num2str(nCompteur));

% Enregistrement de la nouvelle valeur de nCompteur dans les données
d'application de l'objet Figure
setappdata(gcbf,'valeur_de_nCompteur',nCompteur);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION AJOUTER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

3. GUIDATA et GUIHANLDES

Le fichier GUI_GUIDATA_GUIHANDLES.M montre l'utilisation des fonctions **guidata** et **guihandles**. La fonction **guidata** est utilisée pour stocker et retrouver les variables et la fonction **guihandles** est utilisée pour gérer les identifiants des objets.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gui_guidata_guihandles

% Création de l'objet Figure
figure('units','pixels',...
    'position',[250 250 500 500],...
    'color',[0.925 0.913 0.687],...
    'numbertitle','off',...
    'name','[GUI] Utilisation de GUIDATA',...
    'menubar','none',...
    'tag','interface');

% Création de l'objet Uicontrol Pushbutton -
uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.1 0.1 0.1 0.05],...
    'string','-',...
    'callback',@retrancher,...
    'tag','bouton_retrancher');

% Création de l'objet Uicontrol Pushbutton +
uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.3 0.1 0.1 0.05],...
    'string','+',...
    'callback',@ajouter,...
```

```
    'tag','bouton_ajouter');

% Création de l'objet Uicontrol Text résultat
uicontrol('style','text',...
    'units','normalized',...
    'position',[0.1 0.2 0.3 0.05],...
    'string','0',...
    'tag','resultat');

% Génération de la structure contenant les identifiants des objects
graphiques dont la
% propriété Tag a été utilisée.

data=guihandles(gcf);

% D'après les Tag utilisés pour les objets graphiques crées précédemment,
la structure data
% contient les champs suivant :
%   data.interface
%   data.resultat
%   data.bouton_ajouter
%   data.bouton_retrancher
%

% Initialisation de la variable représentant la valeur courante du compteur
nCompteur à 0
% Note : nCompteur est ici un champ de la structure data
data.nCompteur=0;

% Enregistrement de data dans les données d'application de l'objet Figure
guidata(gcf,data)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function retrancher(obj,event)

% Récupération des données stockées dans les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
data=guidata(gcbf);

% Diminution de la valeur de nCompteur
data.nCompteur=data.nCompteur-1;

% Modification de sa propriété String
set(data.resultat,'string',num2str(data.nCompteur));

% Enregistrement des données modifiées dans les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
guidata(gcbf,data)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION RETRANCHER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA SOUS-FONCTION AJOUTER%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ajouter(obj,event)

% Récupération des données stockées dans les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
data=guidata(gcbf);

% Augmentation de la valeur de nCompteur
data.nCompteur=data.nCompteur+1;

% Modification de sa propriété String
set(data.resultat,'string',num2str(data.nCompteur));

% Enregistrement des données modifiées dans les données d'application de
l'objet Figure
% contenant l'objet graphique dont l'action est exécutée (gcbf)
guidata(gcbf,data)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA SOUS-FONCTION AJOUTER%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

4) Les fonctions imbriquées (nested functions)

Le fichier GUI_NESTED_FUNC.M montre l'utilisation des fonctions imbriquées (nested functions). Ce type de fonctions a été intégré dans la version 7 de MATLAB. Elles sont extrêmement souples d'utilisation. Une variable définie dans la fonction parent est uniquement visible dans les fonctions qui y sont imbriquées. L'utilisation de ces variables est donc immédiate, comme pour les variables globales.

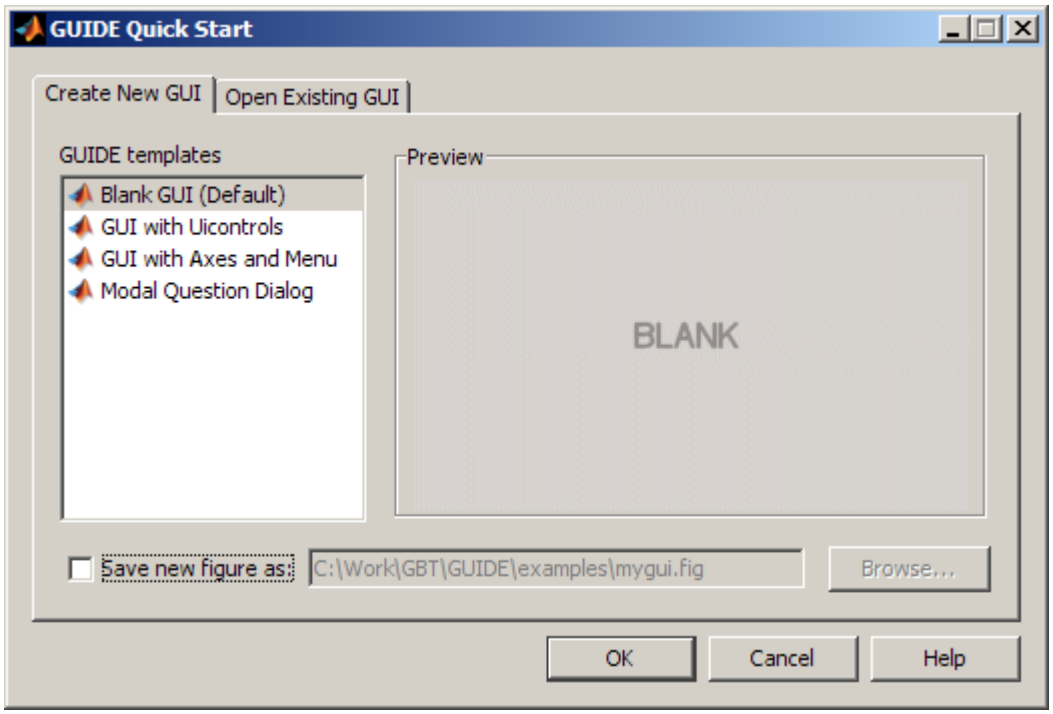
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%DEBUT DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gui_nested_func

% Initialisation de la variable représentant la valeur courante du compteur
nCompteur à 0
nCompteur=0;

% Création de l'objet Figure
handles(1)=figure('units','pixels',...
    'position',[250 250 500 500],...
    'color',[0.925 0.913 0.687],...
    'numbertitle','off',...
    'name','[GUI] Utilisation des variables globales',...
    'menubar','none',...
    'tag','interface');

% Création de l'objet Uicontrol Pushbutton -
handles(2)=uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.1 0.1 0.1 0.05],...
    'string','-',...
    'callback',@retrancher,...
    'tag','bouton-');
```

```
% Création de l'objet Uicontrol Pushbutton +
handles(3)=uicontrol('style','pushbutton',...
    'units','normalized',...
    'position',[0.3 0.1 0.1 0.05],...
    'string','+',...
    'callback',@ajouter,...
    'tag','bouton+');

% Création de l'objet Uicontrol Text résultat
handles(4)=uicontrol('style','text',...
    'units','normalized',...
    'position',[0.1 0.2 0.3 0.05],...
    'string','0',...
    'tag','resultat');

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %DEBUT DE LA FONCTION IMBRIQUEE RETRANCHER%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        function retrancher(obj,event)

        % Diminution de la valeur de nCompteur
        nCompteur=nCompteur-1;

        % Actualisation de la propriété String de l'objet Uicontrol Text
résultat
        set(handles(4),'string',num2str(nCompteur));

         end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %FIN DE LA FONCTION IMBRIQUEE RETRANCHER%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %DEBUT DE LA FONCTION IMBRIQUEE AJOUTER%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        function ajouter(obj,event)

        % Augmentation de la valeur de nCompteur
        nCompteur=nCompteur+1;

        % Actualisation de la propriété String de l'objet Uicontrol Text
résultat
        set(handles(4),'string',num2str(nCompteur));

         end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %FIN DE LA FONCTION IMBRIQUEE AJOUTER%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FIN DE LA FONCTION PRINCIPALE%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

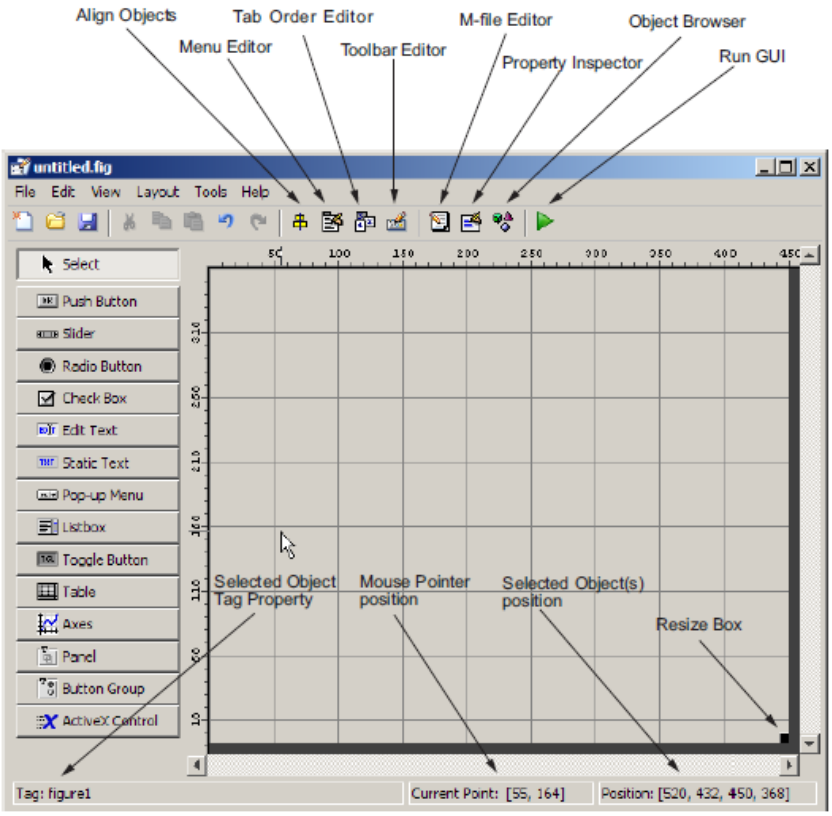
Chapitre 19 : Guide

Dans le terminale de Matlab, faite >> guide

Voici la fenêtre qui apparaît:



Voici l'interface de travail:



Chapitre 20: 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 fig1est 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 partie des sous-menus. L'ajout de menus spécifique se fait par :

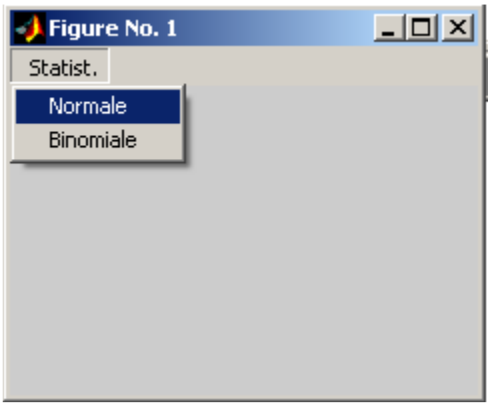
menu1 = uimenu(fig1 , 'label' , ' Statist.');

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.

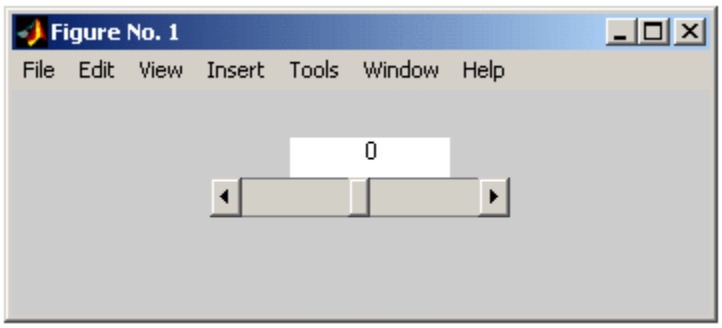
Exemple:

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''))');



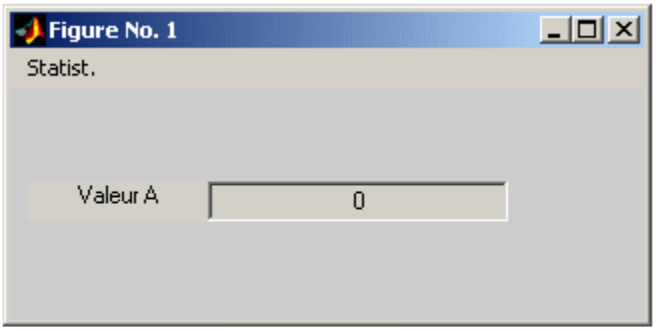
Texte Editable

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.

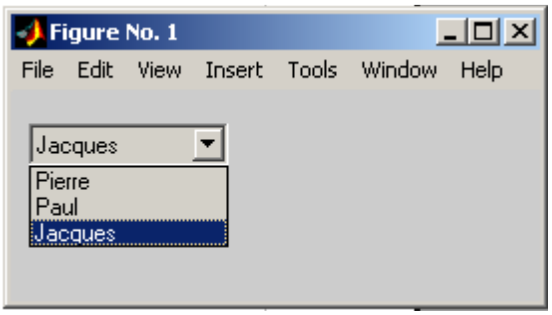
uicontrol (fig1 , 'style' , ' texte' , 'position', [10,50,90,20] , 'string' , 'Valeur A');



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.

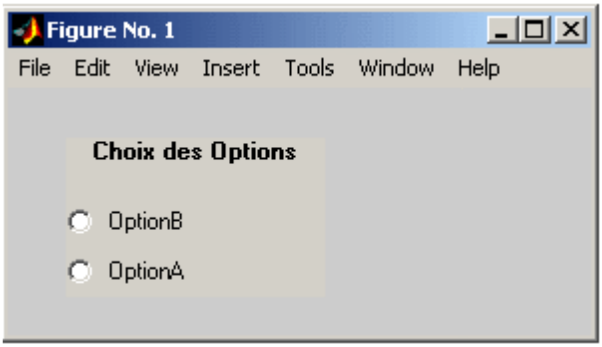
choix1 = uicontrol (gcf , 'Style' , 'popup' , 'String' , 'Pierre|Paul|Jacques' , 'Position' , [10 10 100 80]);



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])
```

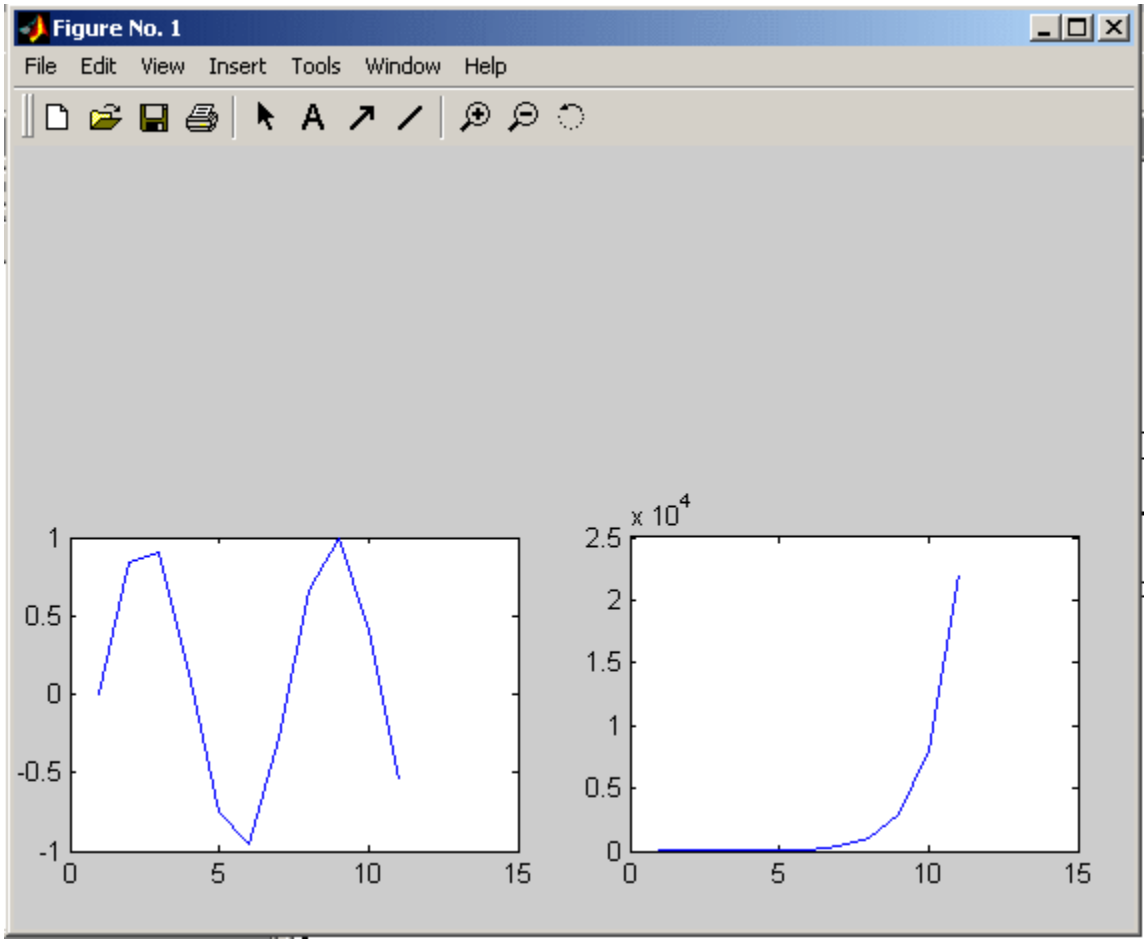
Graphiques

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 (Tag[2]) textStatus.

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

Le code est le suivant :

```
function pushbutton1_Callback(hObjet, evendata, 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)
```

[2] 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?

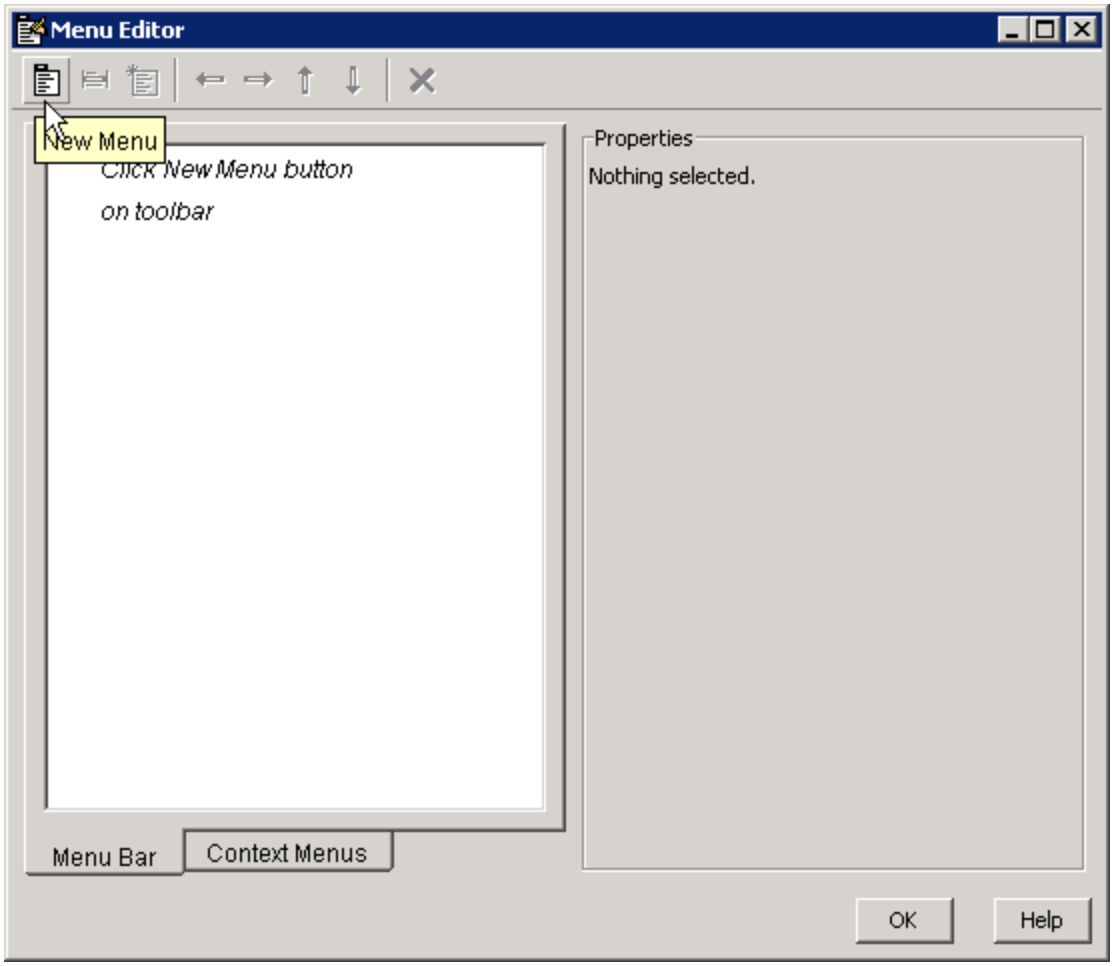
Les commandes de l'action sont les suivantes:

```
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 :



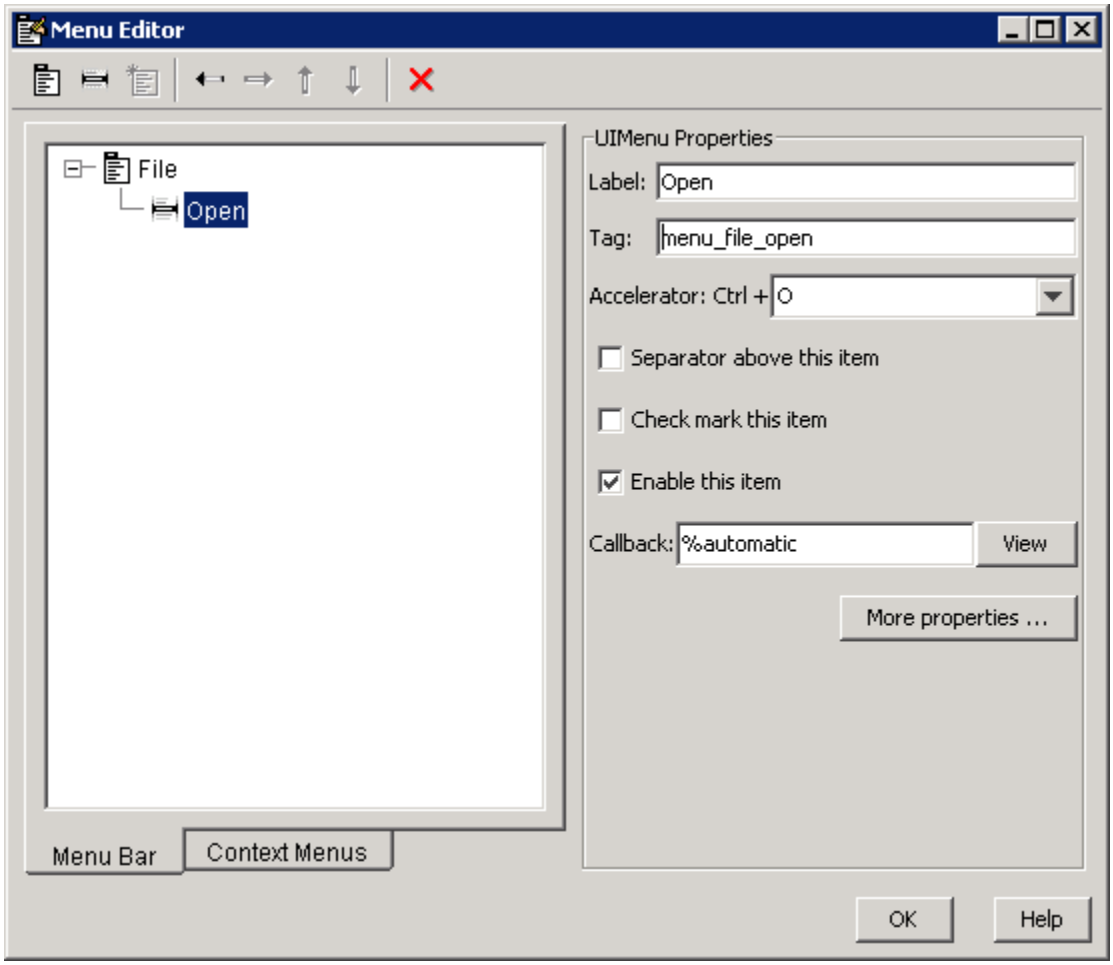
Puis par exemple, faire un menu File est un sous-Menu Open.

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 étiquette[3] (Tag).

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

[3] 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.

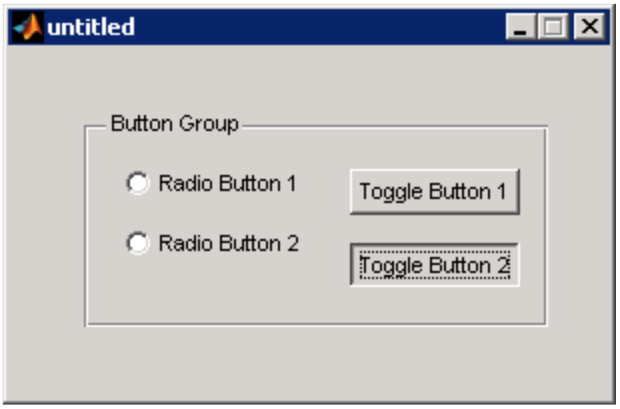
16) Signification de la fonction mygui_OpeningFcn(hObject, eventdata, handles, varargin):

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)[4];
```

17) Comment faire un bouton grouper?



Choisir l'action SelectionChangeFcn, voici le code :

```
function uibuttongroup1_SelectionChangeFcn(hObject,eventdata)
```

[4] Le guidata permet de remettre à jour la structure handles.

```
switch get(eventdata.NewValue,'Tag') % Get Tag of selected object.
        case 'radiobutton1'
% Code for when radiobutton1 is selected.
        case 'radiobutton2'
% 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(figure_handle, '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_edittext1(hObject, eventdata, handles)
mystring = get(hObject,'String');
set(hObject,'UserData',mystring)
string = get(handles.edittext1,'UserData');
```

Chapitre 21 : 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

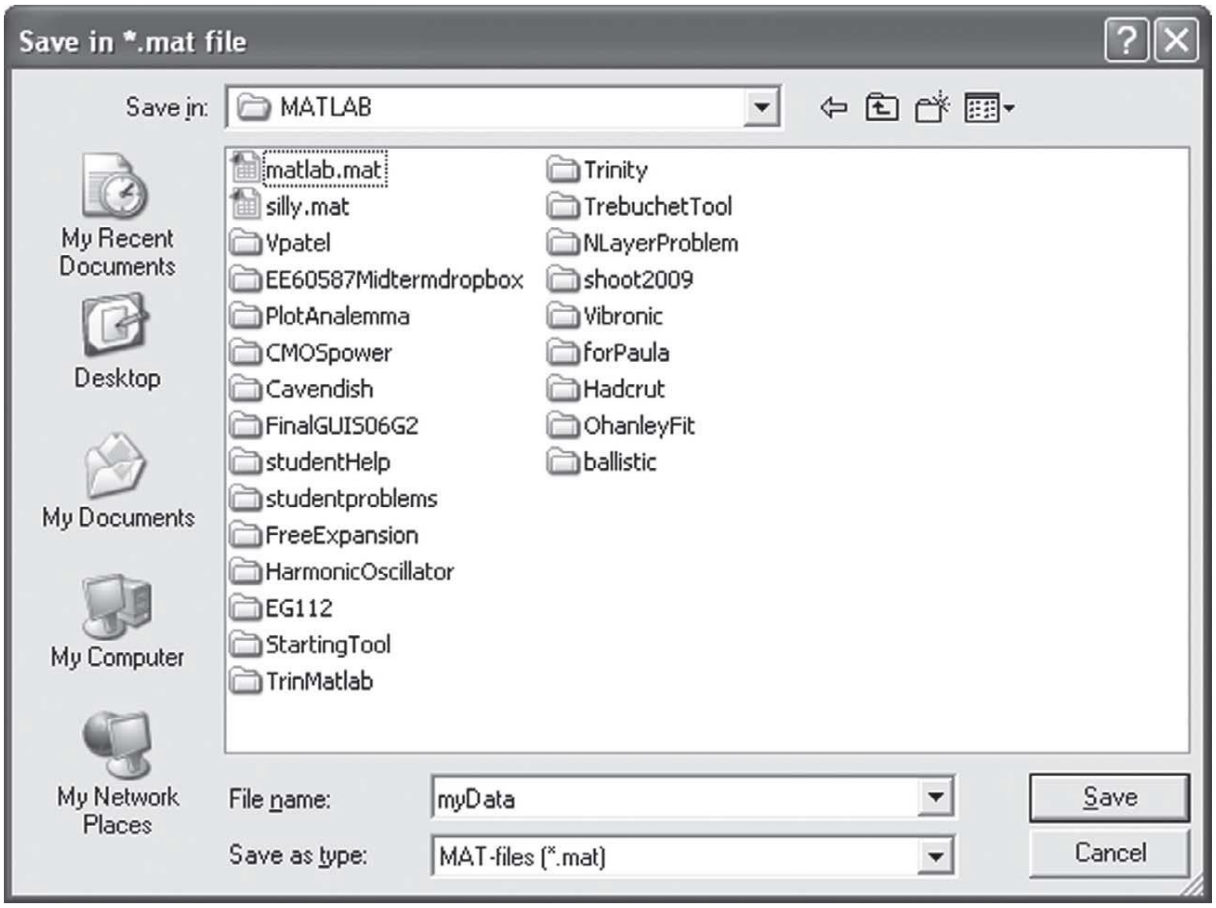
theFile='myData';
load(theFile);
load(theFile, 'x', 'y', 'name')

uiputFile

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

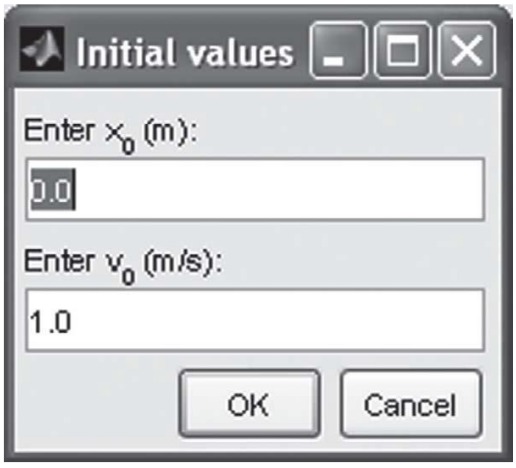
Ou

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



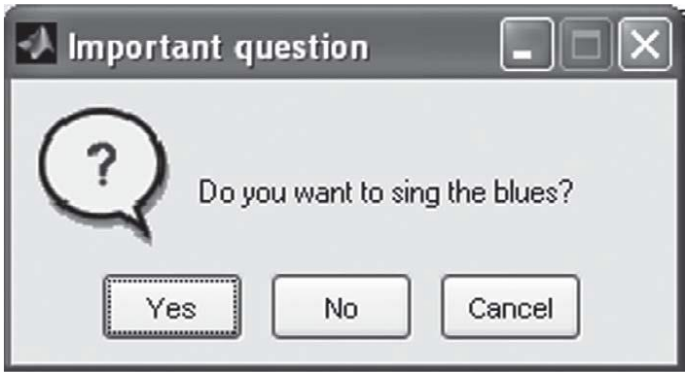
Input dialog

```
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
```



Chapitre 22 : Compléments

I) Les abréviations utiles à connaître:

gcf : get current figure

gca : get current axes

2) Les commandes get et set:

get : propriété sur l'objet

set : changement la propriété de l'objet

Exemples:

```
get(handle, 'PropertyName')
```

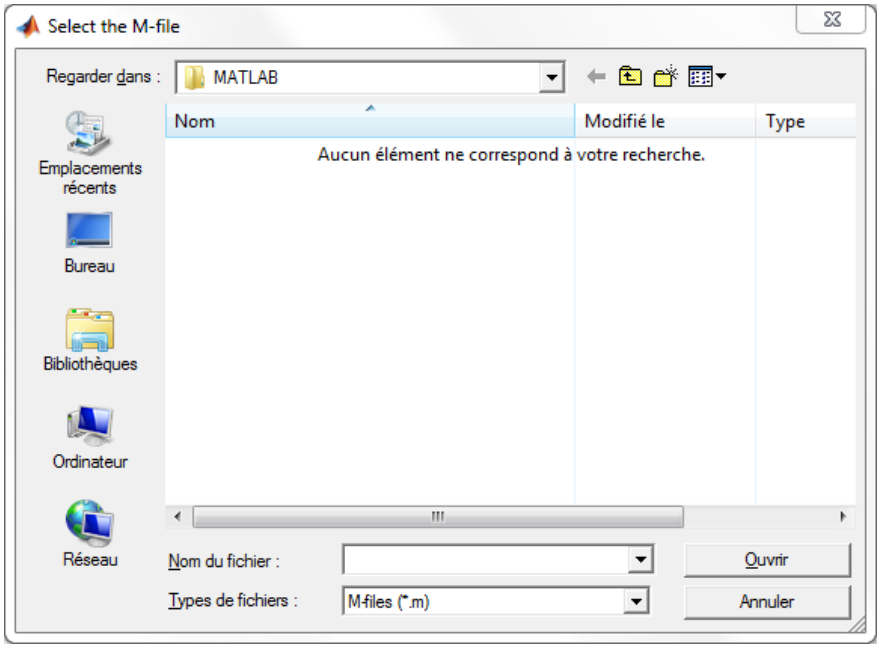
```
set(handles, 'PropertyName', 'PropertyValue')
```

3) uigetfile:

Cette méthode permet d'ouvrir une fenêtre d'ouverture pour ouvrir un fichier.

Plusieurs exemples intéressants:

[FileName,PathName] = uigetfile('*.m','Select the M-file');



[filename, pathname] = ...

uigetfile({'*.m';'*.mdl';'*.mat';'*.*'},'File Selector');

4) uiputfile:

Cette méthode permet d'ouvrir une fenêtre de sauvegarde pour un fichier.

Les syntaxes:

uiputfile

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec)

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec,DialogTitle)

[FileName,PathName,FilterIndex] = uiputfile(FilterSpec,DialogTitle,DefaultName)

Exemple:

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

5) uigetdir:

Cette méthode permet d'ouvrir une fenêtre pour sélectionner un chemin.

Les syntaxes:

uigetdir

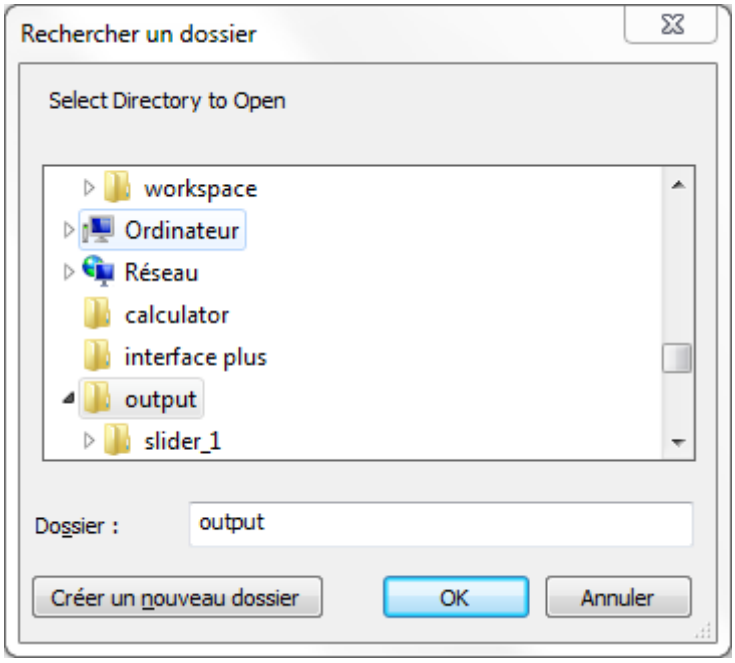
directory_name = uigetdir

directory_name = uigetdir(start_path)

directory_name = uigetdir(start_path,dialog_title)

Exemple:

dname = uigetdir('C:\');



6) fullfile:

Cette méthode permet de concaténer le chemin d'un fichier et le nom du fichier avec son extension.

```
filename = fullfile(pathname ,filename)
filename = C:\Users\admin\Desktop\myfile.txt
```

7) fileparts:

Cette méthode est inverse de la commande "fullfile". Elle donne le chemin, nom et extension d'un fichier.

```
[pathStr,name,extension] = fileparts('C:\myStuff\myfile.txt')
pathStr =
C:\myStuff
name =
 myfile
extension =
.txt
```

8) strcmp:

Permet de comparer deux chaînes de caractères.

```
>> isMatch = strcmp('hello','goodbye')

isMatch =

     0

>> myStr = 'hello';
>> isMatch = strcmp('hello', myStr)

isMatch =

     1
```

9) regexp:

Appel la fonction des expressions régulières.

Syntaxe:

```
[selected_outputs] = regexp(string,expr,outselect).
```

Avec string qui la chaine de caractère, exp est l'expression régulière et outselect est option.

Exemples:

```
mystring = 'My name is Sarah Wait Zaranek';
splitstring = regexp(mystring,'\s','split');
disp(splitstring)
```

Résultat: `'My' 'name' 'is' 'Sarah' 'Wait' 'Zaranek'`

```
mystring = 'My name is Sarah. I love MATLAB! Do you?';
splitstring = regexp(mystring,'(?<=[!.?])\s','split');
disp(splitstring)
```

Résultat: `'My name is Sarah.' 'I love MATLAB!' 'Do you?'`

```
locationNames = {'Bennett Valley' , 'Bishop' , 'Camino', 'Santa Rosa', ...
                 'U.C. Riverside', 'Windsor'};

idx = regexp(locationNames, '[A-Z]');
```

```
shortLabels = cellfun(@(label,idx) label(sort([idx idx+1])),...
    locationNames,idx,'UniformOutput',false);

disp(shortLabels)
```

Résultat: `'BeVa'  'Bi'  'Ca'  'SaRo'  'U.C.Ri'  'Wi'`

```
shortLabels2 = regexp(locationNames, '[A-Z].', 'match');

shortLabelsFinal = cellfun(@(x) [x{:}], shortLabels2,
'UniformOutput',false);
disp(shortLabelsFinal)
```

Résultat: `'BeVa'  'Bi'  'Ca'  'SaRo'  'U.C.Ri'  'Wi'`

```
geneString = 'CC=0/CT=1/TT=5375';

doubleValues1 = regexp(geneString,'(?<=(CC|TT|AA|GG)=)\d+','match');
disp(doubleValues1)
```

Résultat : `'0'  '5375'`

10) varargin:

Si vous avez une fonction de ce type:

```
function myFunction(input1, input2, varargin)
```

Et vous appelez comme suivant:

```
myFunction(56.4, 98.7, 100, 'a string', true)
```

Le résultat de varargin est:

```
varargin =

    [100]    'a string'    [1]
```

Chapitre 23: Exemple de guide

Aller dans le terminal,

Tapez >> guide

Voici la fenêtre qui apparaît :

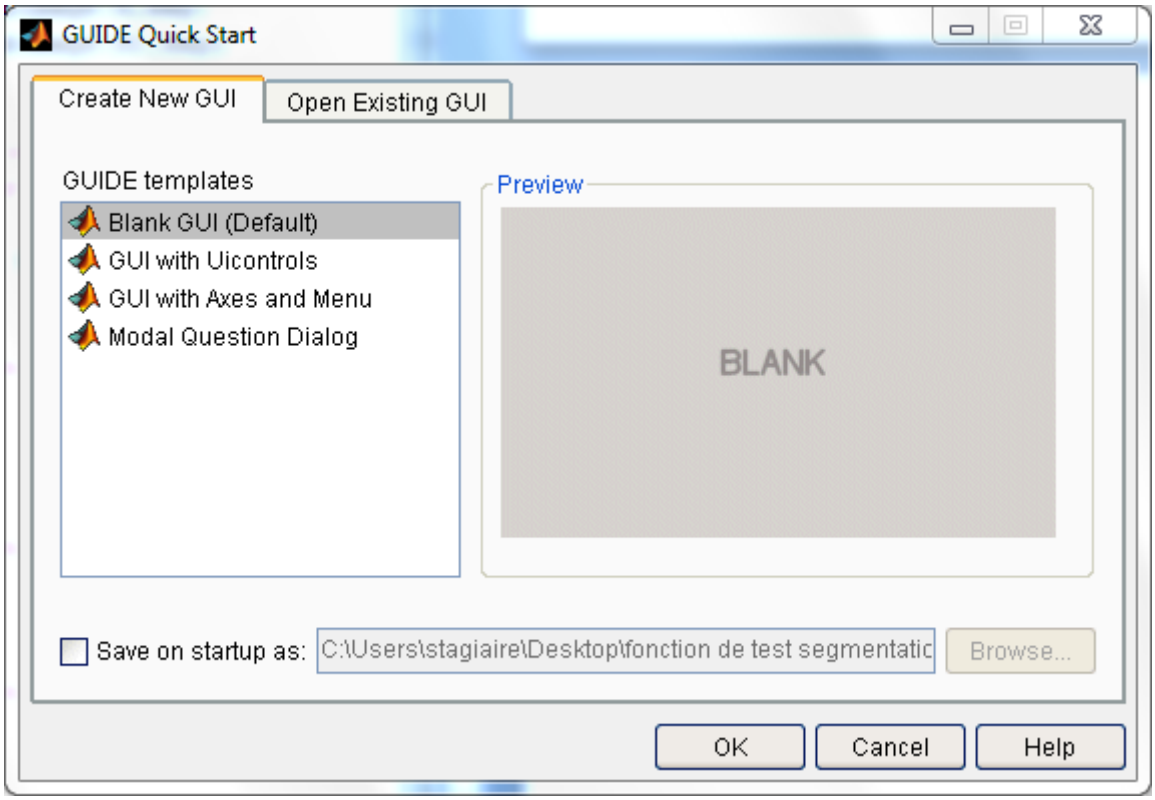


Figure 6 : après guide

Si vous voulez une interface vierge prendre Blank GUI[5].

Vous pouvez changer le chemin de l'enregistrement de l'interface sur *Save on startup as*.

Sur la **figure 7**,

- un fichier .fig (non éditable) contenant les objets graphiques Figure, Axes et Pushbutton
- un fichier .m contenant le code du fonctionnement de l'interface graphique

[5] Pour la suite de ce document, je prendrai Blank GUI.

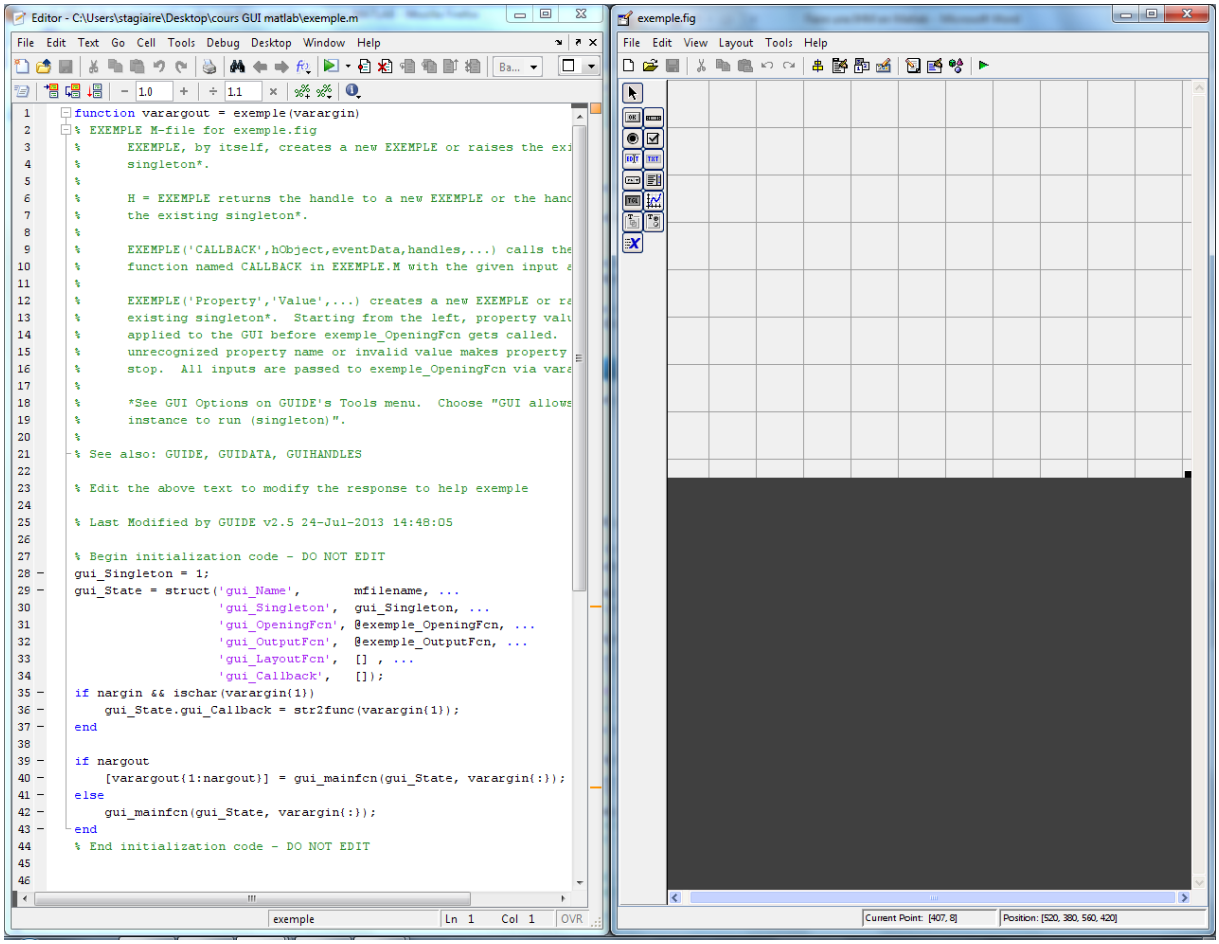
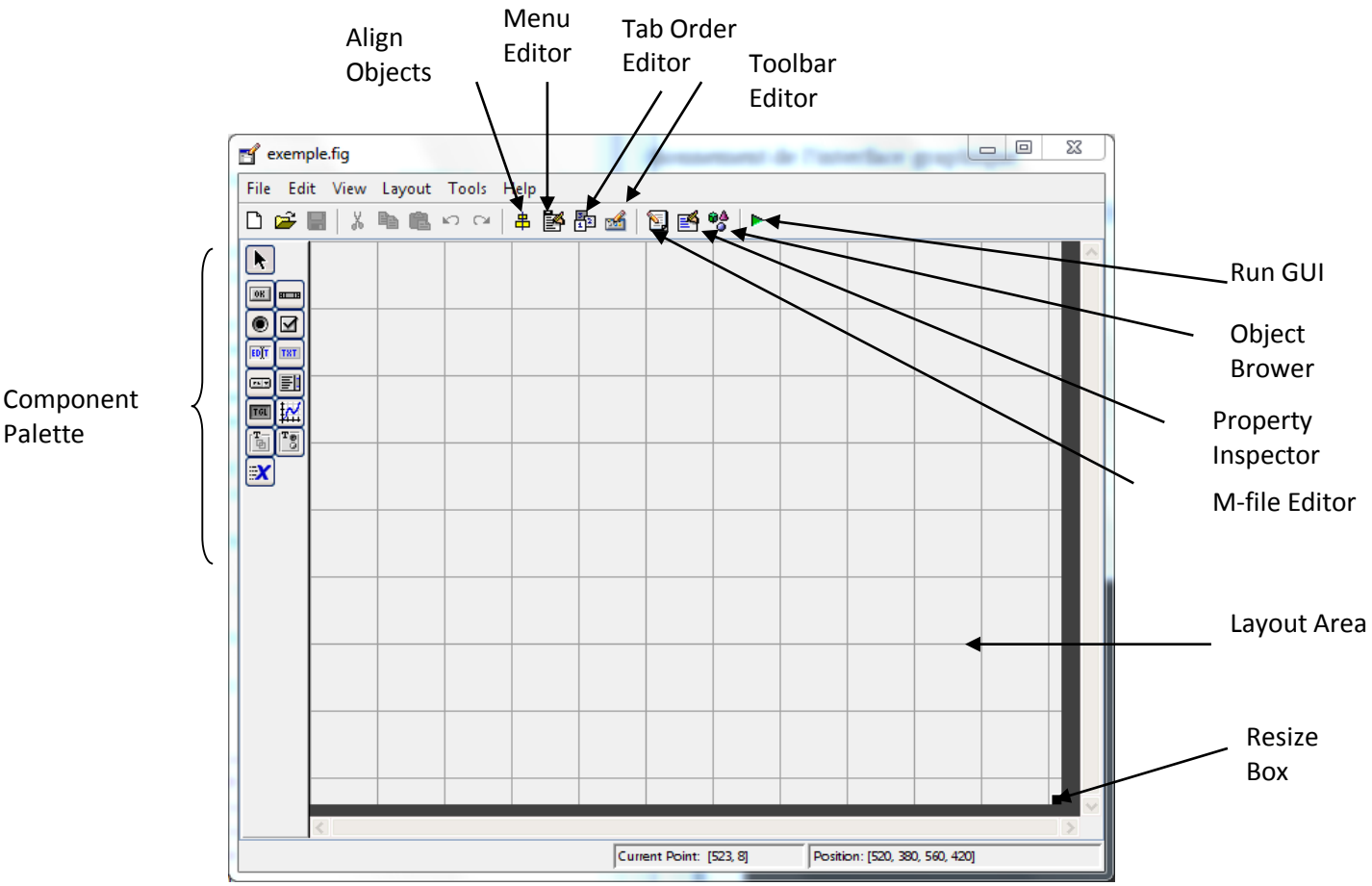


Figure 7 : à gauche le .m et à droite le .fig

Je vais faire une petite présentation du .fig :



Maintenant, nous allons voir le document .m

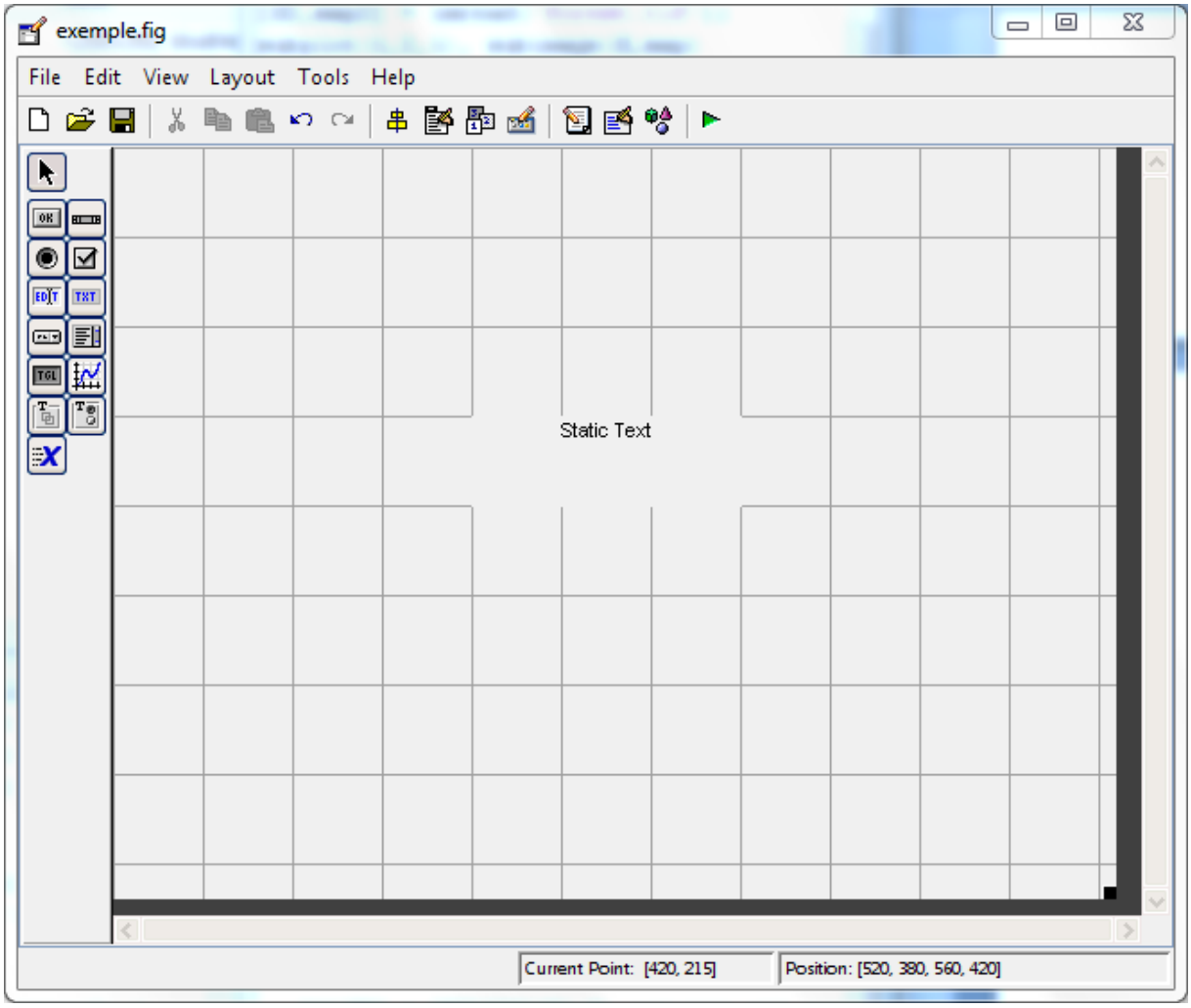
- La fonction "`exemple_OpeningFcn`" permet de faire la gestion de l'ouverture de l'interface.

Le mot guidata(hObject, handles) permet de créer la structure handles.

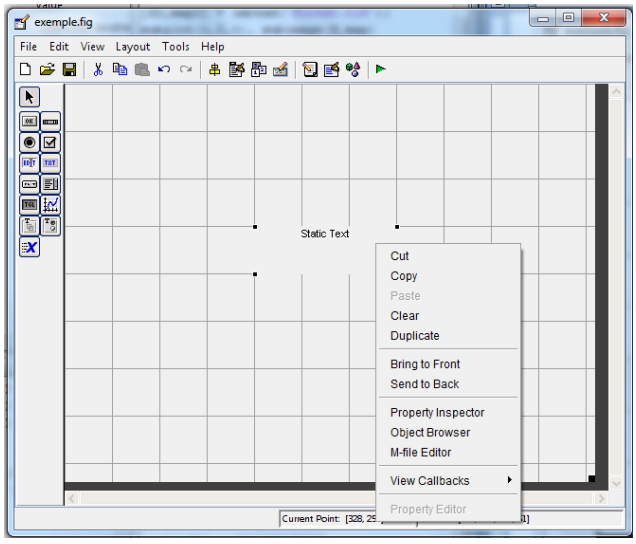
1) Variables globales

>>guide

On va ensuite positionner le cadre text.

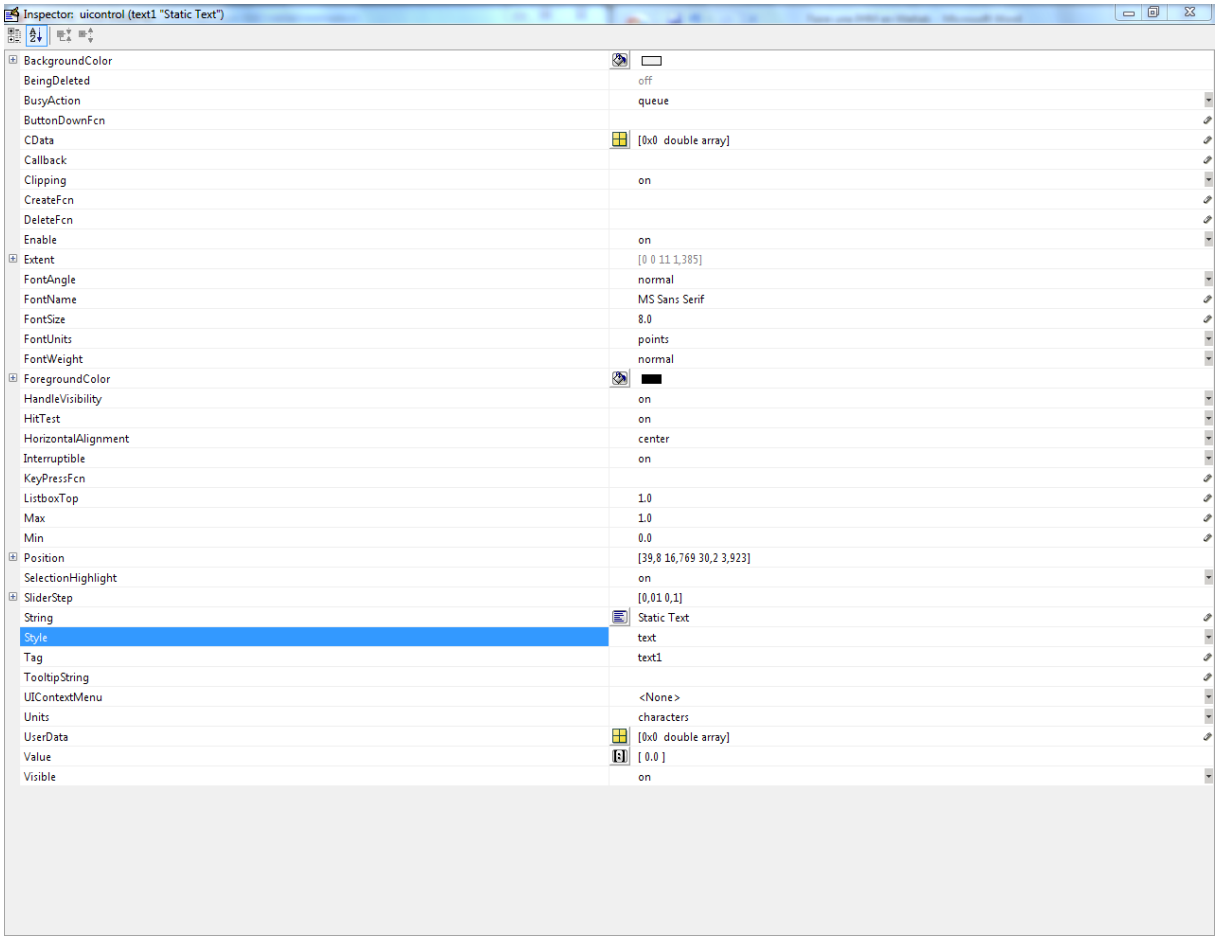


Cliquer bouton droite sur Static Text :

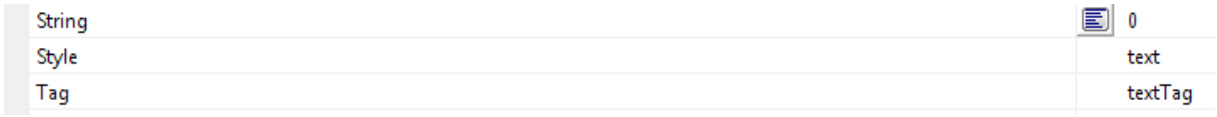


Puis sur Property Inspector : qui est le gestionnaire des propriétés

Voici la fenêtre qui apparaît :



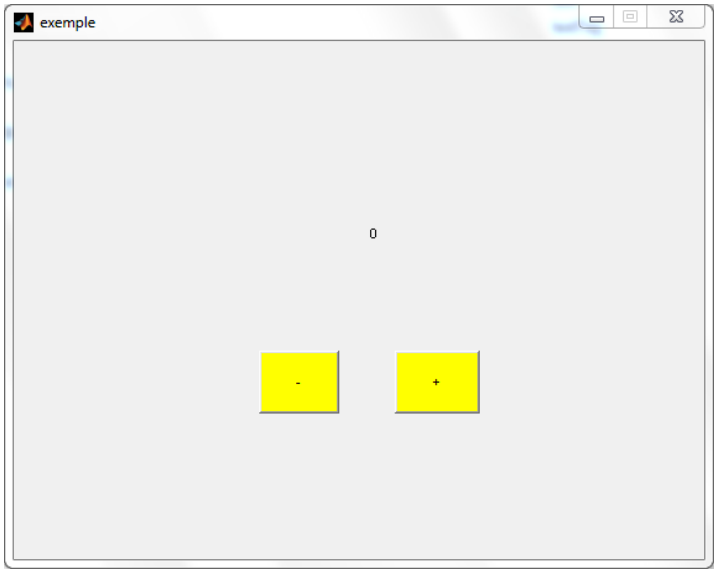
Faisait le changement suivant :



On va ensuite positionner les deux boutons – et +. Nommé ces deux boutons minus et plus.

Puis compiler l'interface graphique (.fig).

Voici l'interface graphique que vous devez obtenir :



Aller maintenant dans le fichier .m :

- En premier, initialisation du compteur à l'ouverture : (←)

```
% --- Executes just before exemple is made visible.
function exemple_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 exemple (see VARARGIN)

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

handles.indice = 0; ←

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

- Implémentation des deux boutons:

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

handles.indice  = handles.indice-1;
set(handles.textTag,'string',num2str(handles.indice));
guidata(hObject,handles);


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

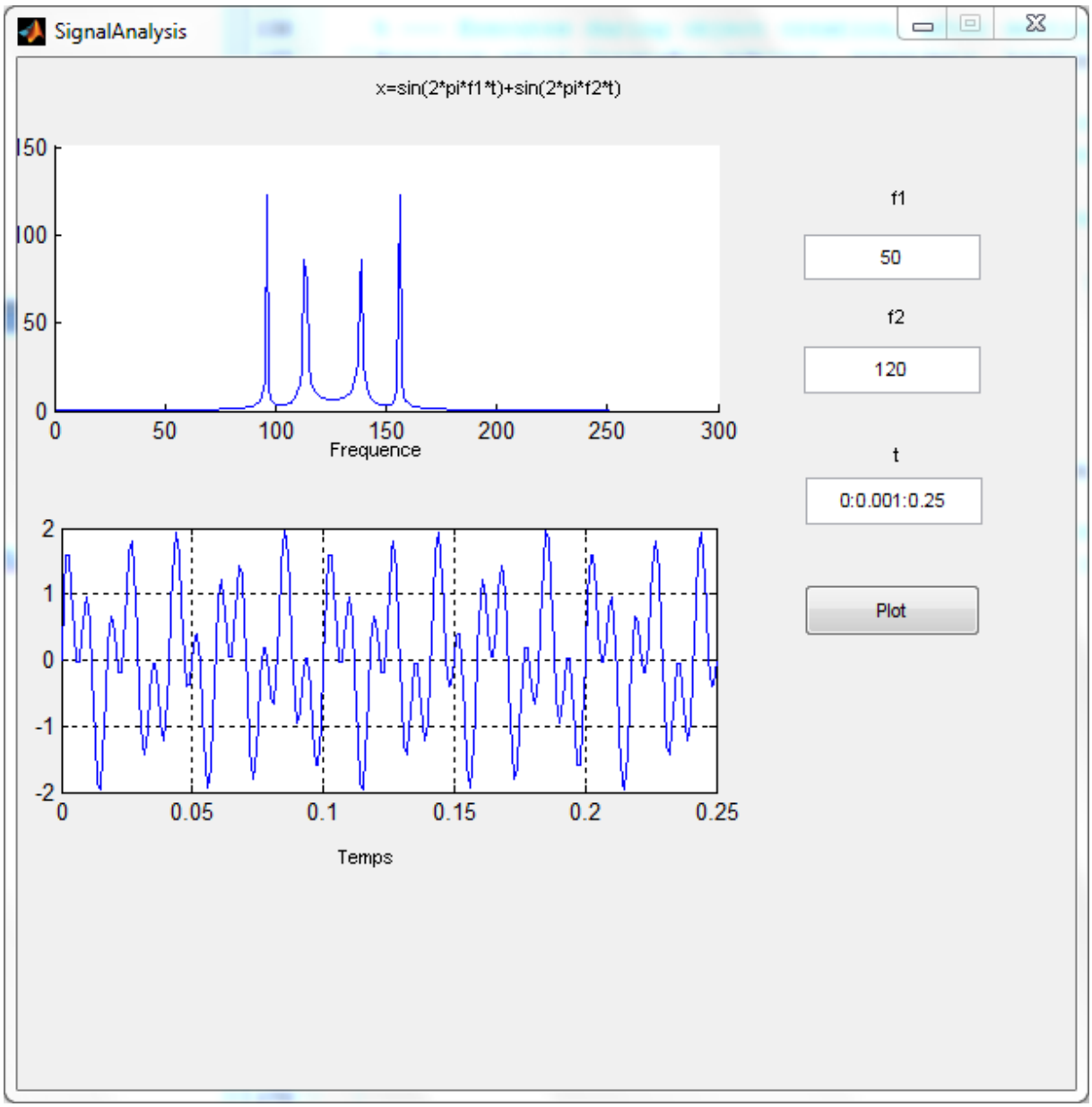
```
handles.indice = handles.indice + 1;% sélectionne le fichier

set(handles.textTag,'string',num2str(handles.indice));

guidata(hObject,handles);
```

Projet IHM avec GUIDE

Signal analyse:

L'objectif est de faire une interface graphique pour une certaine équation donnée et d'afficher sa transformée de Fourier et le déplacement en fonction du temps.

Le résultat souhaiter est le suivant :



1) Reproduisez cette interface ; (avec guide)
2) Premier travail est d'initialiser les variables de saisie (f1, f2, et t) dans le .m

Indices :
Penser à l'endroit où le programme s'ouvre « `OpeningFcn` »

```
set(handles.edit1,'String', num2str(50))
set(handles.edit2,'String', num2str(120))
set(handles.edit3,'String', '0:0.001:0.25')
```

3) Créer la fonction computing

Elle aura pour syntaxe : `function x = computing(f1, f2, t)`

4) Aller sur votre .fig puis sur le bouton « Plot », créer la méthode de dialogue entre fonction (clique droite puis sur View Callbacks puis Callbacks)
5) Dans cette nouvelle fonction,
   - Récupérer les valeurs de f1 et f2 puis les convertir en double
   - Récupérer la valeur temps puis convertir en entier

Indice :

Utiliser la fonction get(nom de l'objet, 'String'), str2double et str2num

- Ensuite appeler la fonction calcul `y = computing(f1, f2, t) ;`
- Puis la transformée de Fourier (commande fft(y))
- Faire les différentes commandes d'affichage

Indice :

Utiliser set(nom du plot par exemple, 'parent', handles.nom de l'objet graphique)
Utiliser plot, et grid on

REPONSE :

```
function varargout = SignalAnalysis(varargin)
% SIGNALANALYSIS M-file for SignalAnalysis.fig
%      SIGNALANALYSIS, by itself, creates a new SIGNALANALYSIS or raises
the existing
%      singleton*.
%
%      H = SIGNALANALYSIS returns the handle to a new SIGNALANALYSIS or the
handle to
%      the existing singleton*.
%
%      SIGNALANALYSIS('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in SIGNALANALYSIS.M with the given input
arguments.
%
%      SIGNALANALYSIS('Property','Value',...) creates a new SIGNALANALYSIS
or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before SignalAnalysis_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to SignalAnalysis_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help SignalAnalysis

% Last Modified by GUIDE v2.5 24-Jul-2013 16:34:07

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SignalAnalysis_OpeningFcn, ...
                   'gui_OutputFcn',  @SignalAnalysis_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 SignalAnalysis is made visible.
function SignalAnalysis_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 SignalAnalysis (see VARARGIN)

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


set(handles.edit1,'String', num2str(50))
set(handles.edit2,'String', num2str(120))
set(handles.edit3,'String', '0:0.001:0.25')
% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = SignalAnalysis_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;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (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,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (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,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a
double


% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (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,'String') returns contents of edit3 as text
%        str2double(get(hObject,'String')) returns contents of edit3 as a
double


% --- Executes during object creation, after setting all properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit 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


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

f1 = str2double(get(handles.edit1,'String'));
f2 = str2double(get(handles.edit2,'String'));
t = str2num(get(handles.edit3,'String') );

y = computing(f1, f2, t);
```

```
Y = fft(y);

hfft = plot(fftshift(abs(double(Y))));
grid on
set(hfft,'parent',handles.axes1);

himg = plot(t ,y );
grid on
set(himg,'parent',handles.axes2);

function x = computing(f1, f2, t)
x= sin(2*pi*f1*t)+sin(2*pi*f2*t);
```

Interface Image

Nous allons effectuer un petit projet graphique pour voir la maîtrise de la GUI sous Guide de Matlab.

But : De faire une interface graphique (figure 1) pour visualiser des images dans un répertoire source et faires avancés les images et dans une deuxième étape d'afficher des informations de cette images.

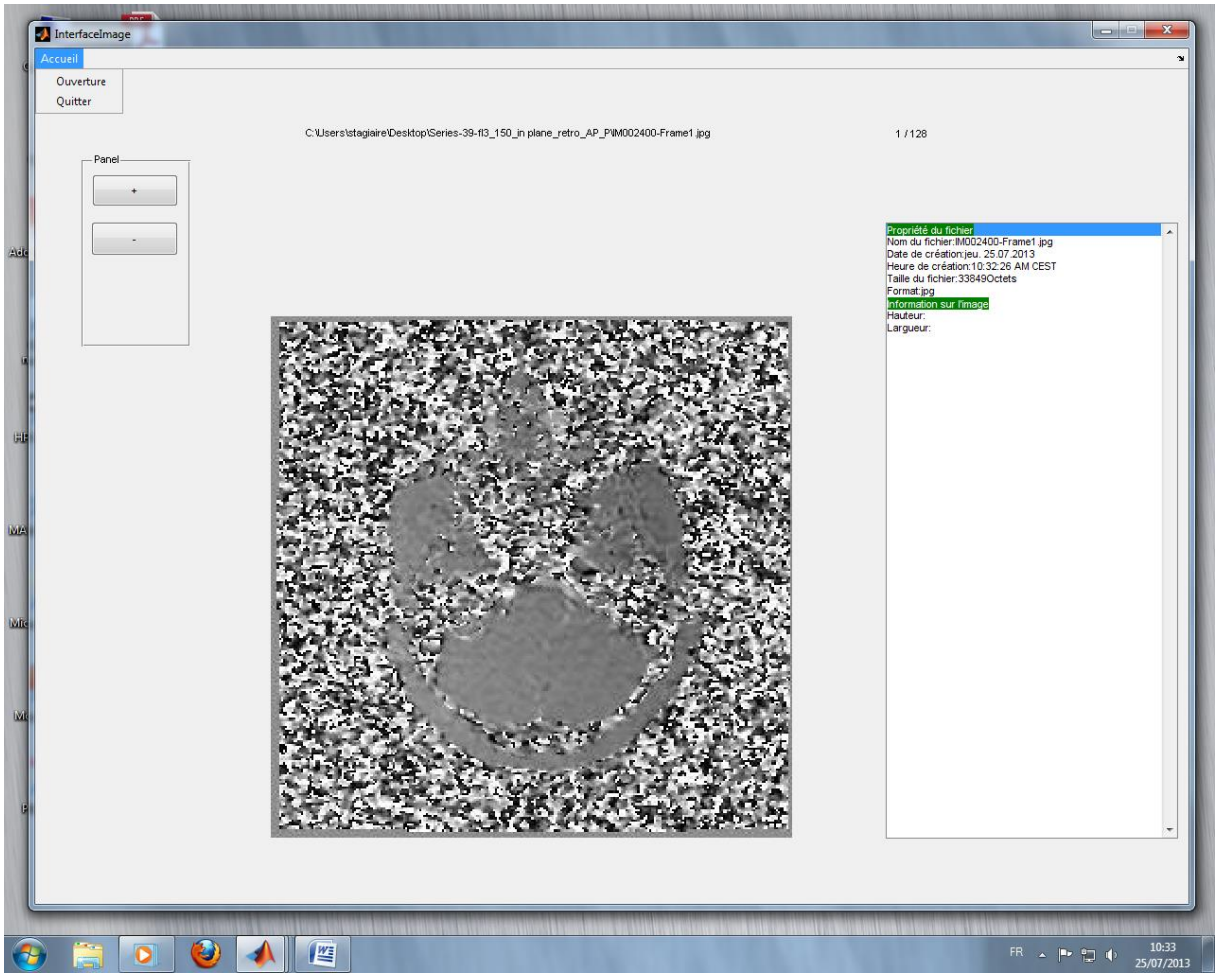


Figure 1 : L'objectif

Commençons ce TP :

I) Initialisations et créations des objets par guides :

1) Lancer Matlab
2) Dans le terminal, marquer guide
3) Choisir Blank GUI puis changer le chemin et le nom (ex : InterfaceImage)
4) Aller dans le point .fig
5) Reproduisez la figure 2

Astuce : Cliquez bouton droite sur l'objet pour avoir sa propriété (property Inpector) et penser aux boutons de faire la méthode callback.

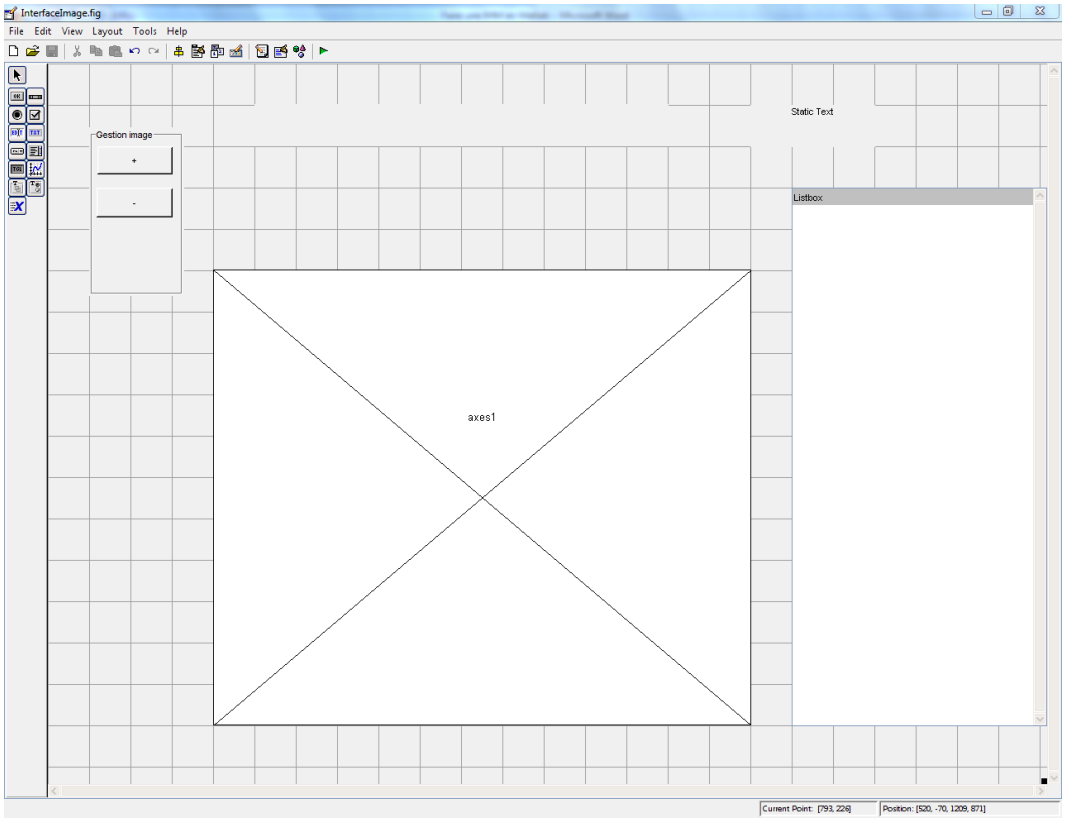


Figure 2 : .fig

Vous devez avoir 2 boutons (+ et -), 1 uiPanel (Gestion Image), 2 labels, 1 fenêtre d'affichage (axes1), 1 Listbox.

6) Faisons maintenant un Menu
   a. Cliquez soit avec le raccourci « Menu Editor » ou aller sur Tools puis Menu editor (Figure 3)

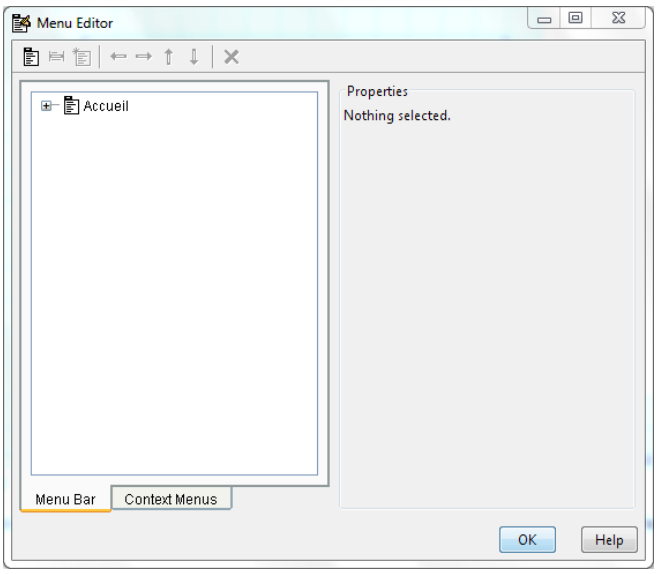


Figure 3 : Fenêtre de Menu Editor

Reproduisez la figure 4

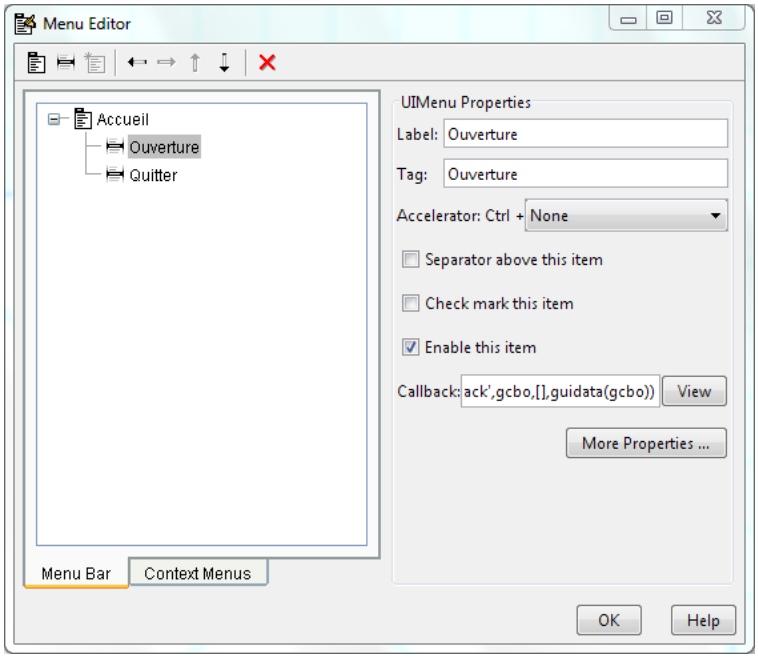


Figure 4 : Fenêtre de Menu Editor à reproduire

7) Lancez run pour générer le code

II) Gestion du code .m

1) Dans la fonction `OpeningFcn(hObject, eventdata, handles, varargin)`, mettez le code suivant qui sont des variables initialisations.

```
handles.output = hObject;
handles.indice =0; % indice à 0 pour l"incrémentation
handles.boolean = false;
```

2) Dans la fonction `Ouverture_Callback(hObject, eventdata, handles)`, Vous devez utilisez l'instruction suivante :

- `rep = uigetdir(matlabroot,'Sélectionner un répertoire d''image');`

Puis créer la méthode de lecture des chemins de répertoire nommée par exemple `[fS, noun] = readdir(rep);% donne le chemin`
Elle retournera cette méthode une structure de tableau fS (les chemins des images) et noun (le nom des images).
Après écrivez cette ligne pour récupérer le nombre de fichier dans un répertoire `[garb noofiles] = size(fS);% la taille du nombre de fichier`

Pour gérer l'affichage et gestion de la Listbox, voici les deux fonctions que vous allez créer.
`affichage(hObject,handles)`
`gestionListbox(hObject,handles)`

Astuce : Pensez à utiliser le mot handles pour faire des variables réutilisable dans le code et guidata(hObject,handles).

3) Dans la fonction `Quitter_Callback(hObject, eventdata, handles)`
Reproduisez ce code et l'étudier.

```
button = questdlg('Ready to quit?', ...
            'Exit Dialog','Yes','No','No');
        switch button
            case 'Yes',
                disp('Exiting MATLAB');
                %Save variables to matlab.mat
                close all;
            case 'No',
                quit cancel;
        end
```

4) La function readdir qui devra prendre le chemin du répertoire et ressortir une structure des chemins+noms des fichiers et que les noms de fichiers.

Voici le code :

```
function [fileSorted, fileSorted2] = readdir(dirName)


if dirName==0
    fileSorted=[];
    disp(' Empty directory ');
else
    D=dir(dirName);
    [nooffilesf garb]=size(D);
    fileX = [];
    for i=3:nooffilesf
        % this assumes that the first and second are . and ..
        % uncomment the if statement below if you want to select only
        % filenames starting with some characters say 'im' in this example
        % similarly we can have a condition to check if the files end in
        % say .dcm
            fileX(i-2).name = D(i).name;
    end

    [garb nooffiles]=size(fileX);

    fileSorted=[];
    fileSorted2=[];
    for i=1:nooffiles
        fileSorted(i).name = strcat(dirName,'\',fileX(i).name);%Path + Name
filter
        fileSorted2(i).name = strcat(fileX(i).name); % Name file
    end
end
```

5) Nous allons effectuée la fonction affichage (`affichage(hObject,handles))`

Lire la matrice image (appeler l'image en utilisant `char(handles.fS2(handles.indice).name)`)

- Affichez l'image (utiliser imshow et axis image)
- Puis l'appeler set pour l'afficher

6) Gestion du bouton plus :

Comprenez ce code et faites de même pour le bouton moins.

```
% --- Executes on button press in pushbuttonPlus.
function pushbuttonPlus_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonPlus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.boolean == true % ouverture de l'incrémentation

handles.indice = handles.indice + 1;% sélectionne le fichier
     if(handles.indice > handles.noofiles)
          handles.indice = 1;


     end
  affichage(hObject, handles);
  gestionListbox(hObject,handles)
end

    guidata(hObject,handles);
```

7) Bonus: Gestions de la listeBox

a. Reproduisez la figure 1

Indices:

- Pour la gestion de couleur de texte utiliser les deux instructions

```
NewInformationFile = 'Propriété du fichier';
      NewColorFile = sprintf('<HTML><BODY bgcolor="green"
text="white">%s', NewInformationFile);
```

- Une des solutions élégeantes pour afficher la date, heure et la taille du fichier passé par java
- Utiliser l'instruction imfinfo
- Pensez à faire une structure pour afficher l'ensemble des éléments que nous allons afficher dans la listbox.
- Puis pensez à l'instruction set

Bonne chance et la correction se trouve en annexe.

Réponse :

```
function varargout = InterfaceImage(varargin)
% INTERFACEIMAGE M-file for InterfaceImage.fig
%      INTERFACEIMAGE, by itself, creates a new INTERFACEIMAGE or raises
the existing
%      singleton*.
%
%      H = INTERFACEIMAGE returns the handle to a new INTERFACEIMAGE or the
handle to
%      the existing singleton*.
%
%      INTERFACEIMAGE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in INTERFACEIMAGE.M with the given input
arguments.
%
%      INTERFACEIMAGE('Property','Value',...) creates a new INTERFACEIMAGE
or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before InterfaceImage_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to InterfaceImage_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help InterfaceImage

% Last Modified by GUIDE v2.5 25-Jul-2013 09:27:15

% By Dimitri PIANETA


% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @InterfaceImage_OpeningFcn, ...
                   'gui_OutputFcn',  @InterfaceImage_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 InterfaceImage is made visible.
function InterfaceImage_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 InterfaceImage (see VARARGIN)

% Choose default command line output for InterfaceImage
handles.output = hObject;
handles.indice =0; % indice à 0 pour l"incrémentation
handles.boolean = false;


% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are returned to the command line.
function varargout = InterfaceImage_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 selection change in listbox1.
function listbox1_Callback(hObject, eventdata, handles)
% hObject    handle to listbox1 (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 listbox1 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from listbox1


% --- Executes during object creation, after setting all properties.
function listbox1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox 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

%% Gestion de l'affichage
function affichage(hObject,handles)
```

```
Y = imread(char(handles.fS2(handles.indice).name));

handles.himg =double(Y);
colormap('default');


himg=imshow(Y);
handles.image = himg;


axis image

  handles.countValue = [handles.indice,'\',handles.noofiles];

  %disp(handles.countValue);

set(himg,'parent',handles.axes1);
set(handles.textPath, 'String',char(handles.fS2(handles.indice).name));


set(handles.text2, 'String',sprintf(' %d /
%d',round(handles.indice),round(handles.noofiles)));


guidata(hObject,handles);

%% Gestion de la listbox
function gestionListbox(hObject, handles)
        NewInformationFile = 'Propriété du fichier';
        NewColorFile = sprintf('<HTML><BODY bgcolor="green"
text="white">%s', NewInformationFile);
        NewInformationImage = 'Information sur l''image';
        NewColorFile2 = sprintf('<HTML><BODY bgcolor="green"
text="white">%s', NewInformationImage);

        myFile = java.io.File(handles.fS2(handles.indice).name);
        flen = length(myFile);
        flenDateTime = java.util.Date(myFile.lastModified());

        % E : Day in week
        % y : Year in four digits
        % M: Month in year
        % d : Day in month
        ft = java.text.SimpleDateFormat('E dd.MM.yyyy');
        uFormat= ft.format(flenDateTime);

        % h : Hour in A.M/P.M
        % m: Minute in hour
        % a : A.M./P.M. marker
        % z: Time Zone example CEST : Central European Summer Time or CET:
        % Central European Time
        %
        ft2 = java.text.SimpleDateFormat('hh:mm:ss a zzz');
        uFormat2= ft2.format(flenDateTime);


    info = imfinfo(char(handles.fS2(handles.indice).name));
```

```
      s = struct('f',{NewColorFile,['Nom du
fichier:',handles.noun(handles.indice).name ],['Date de création:',char(
uFormat) ],['Heure de création:',char( uFormat2) ],...
          ['Taille du
fichier:',num2str(flen),'Octets'],['Format:',info.Format], NewColorFile2,
['Hauteur:',info.Width],['Largueur:',info.Height]} );

%info.MaxSampleValue


set(handles.listbox1, 'String', char(s.f));

%% Gestion de l'ouverture donc du bouton ouverture répertoire
function Ouverture_Callback(hObject, eventdata, handles)
% hObject    handle to Ouverture (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
 rep = uigetdir(matlabroot,'Sélectionner un répertoire d''image');
[fS, noun] = readdir(rep);% donne le chemin
[garb noofiles] = size(fS);% la taille du nombre de fichier

handles.fS2 = fS;
handles.noun = noun;
handles.noofiles = noofiles;
handles.boolean = true;
handles.indice = 1;

guidata(hObject,handles);
affichage(hObject,handles)
gestionListbox(hObject,handles)


%% Bouton quitter
function Quitter_Callback(hObject, eventdata, handles)
% hObject    handle to Quitter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
button = questdlg('Ready to quit?', ...
            'Exit Dialog','Yes','No','No');
        switch button
            case 'Yes',
                disp('Exiting MATLAB');
                %Save variables to matlab.mat
                close all;
            case 'No',
                quit cancel;
        end

%% Gestion des chemins
%
function [fileSorted, fileSorted2] = readdir(dirName)


if dirName==0
    fileSorted=[];
    disp(' Empty directory ');
else
    D=dir(dirName);
    [nooffilesf garb]=size(D);
```

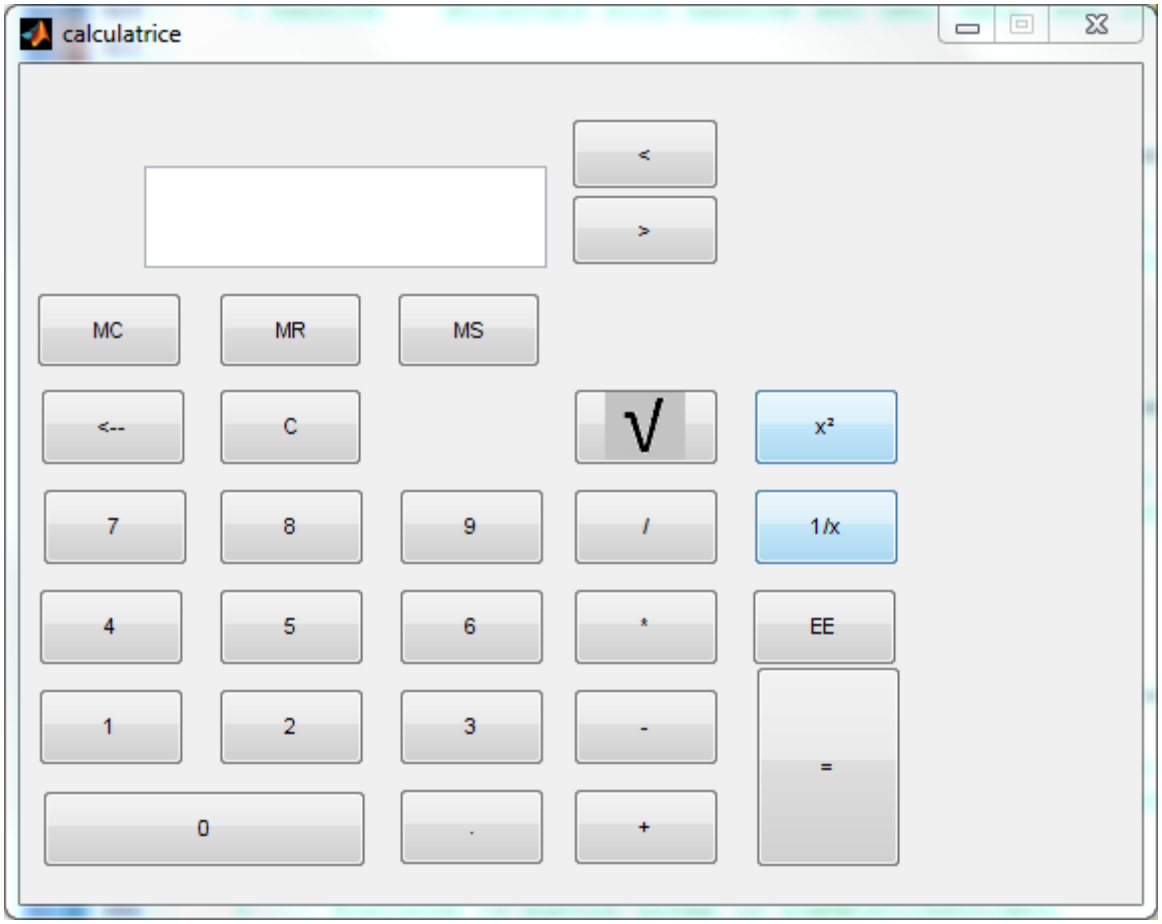
```
    fileX = [];
    for i=3:nooffilesf
        % this assumes that the first and second are . and ..
        % uncomment the if statement below if you want to select only
        % filenames starting with some characters say 'im' in this example
        % similarly we can have a condition to check if the files end in
        % say .dcm
            fileX(i-2).name = D(i).name;
    end

    [garb nooffiles]=size(fileX);

    fileSorted=[];
    fileSorted2=[];
    for i=1:nooffiles
        fileSorted(i).name = strcat(dirName,'\',fileX(i).name);%Path + Name
filter
        fileSorted2(i).name = strcat(fileX(i).name); % Name file
    end
end


% --- Executes on button press in pushbuttonPlus.
function pushbuttonPlus_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonPlus (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
if handles.boolean == true % ouverture de l'incrémentation

handles.indice = handles.indice + 1;% sélectionne le fichier
     if(handles.indice > handles.noofiles)
          handles.indice = 1;

     end
  affichage(hObject, handles);
  gestionListbox(hObject,handles)
end

    guidata(hObject,handles);


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

if handles.boolean == true % ouverture de l'incrémentation

handles.indice = handles.indice - 1;% sélectionne le fichier
     if(handles.indice <1)
          handles.indice = handles.noofiles;

     end

     affichage(hObject, handles);
     gestionListbox(hObject,handles)

end
```

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

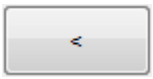
Calculatrice

Le but ce premier TP est d'effectué une calculatrice en GUI en s'aidant avec guide de Matlab.

Avec différentes touches comme l'interface graphique de mon application:



Je vais maintenant décrire l'ensemble des touches de cette interface graphique.

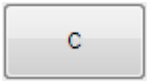


Permet de remonter dans l'historique des opérations que nous avons effectué

Permet de descendre dans l'historique des opérations que nous avons effectué

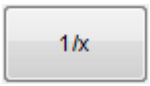
Rappel mémoire

Effacer un par un les chiffres

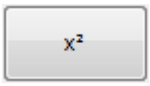


Touche "clear"

Racine carré

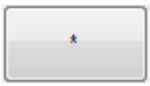


Touche 1 divisé par un nombre x.



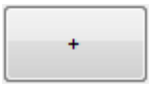
L'opération x au carré

Division



Multiplication

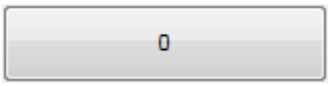
Soustraction

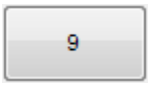


Addition

Égale

Puissance de 10



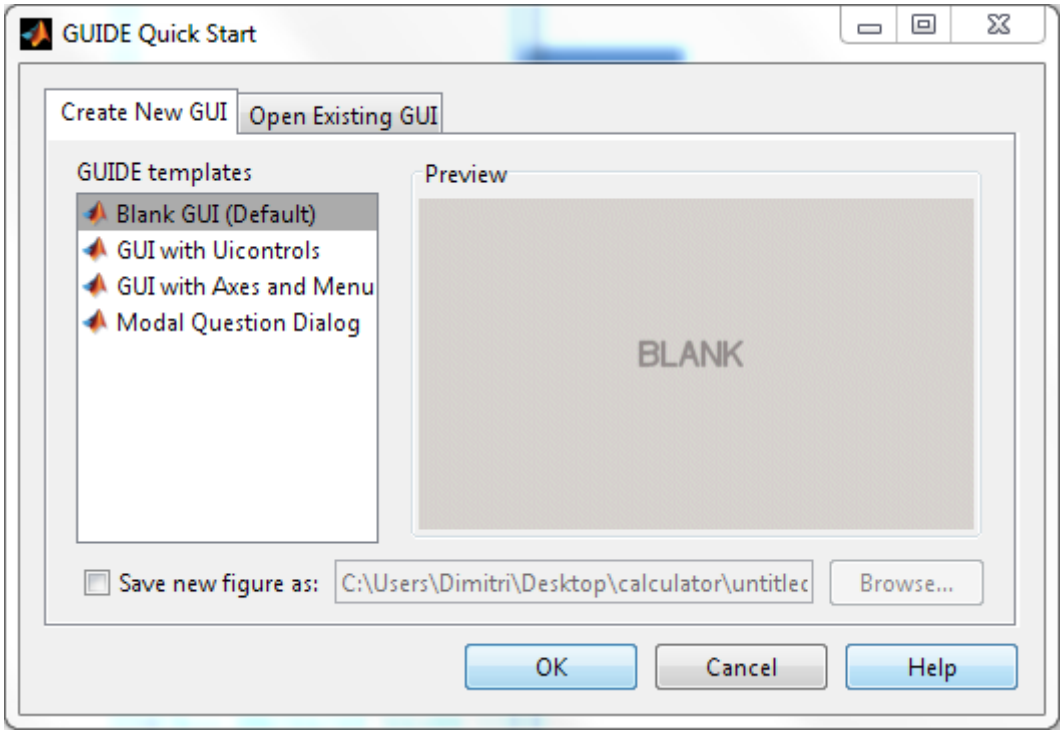
à


Les chiffres de 0 à 9

Étape 0: initialisation

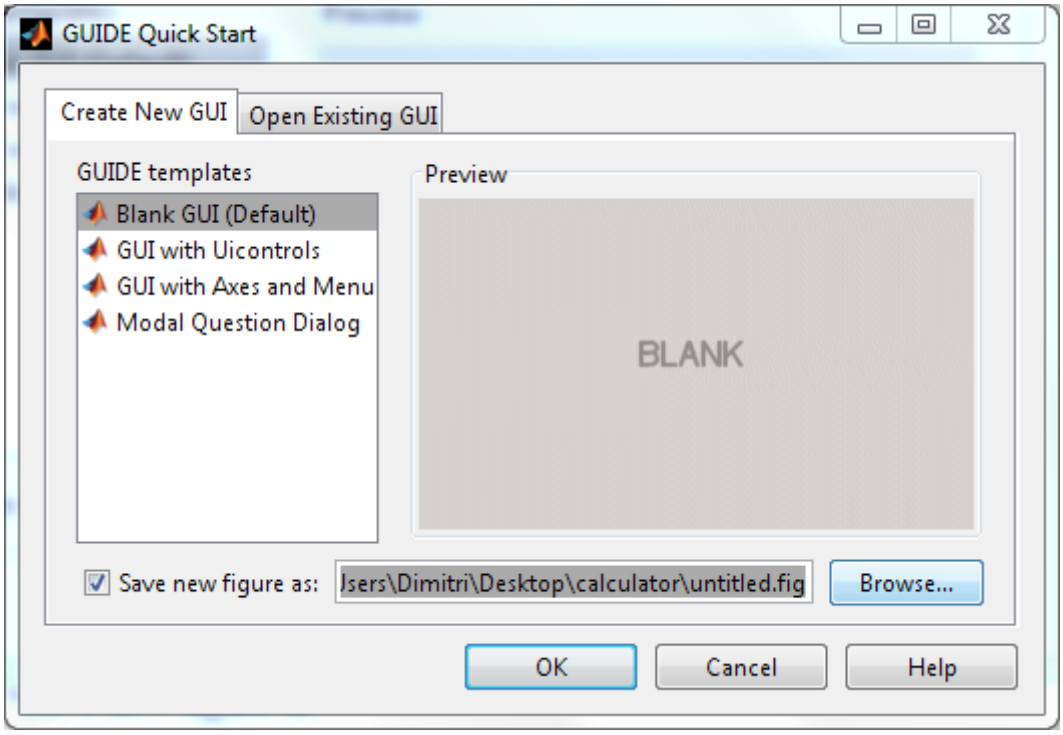
Faite dans le terminale guide

Puis aller sur Blank GUI(Default)



Puis cocher "Save new figure as"

```
>> guide
```

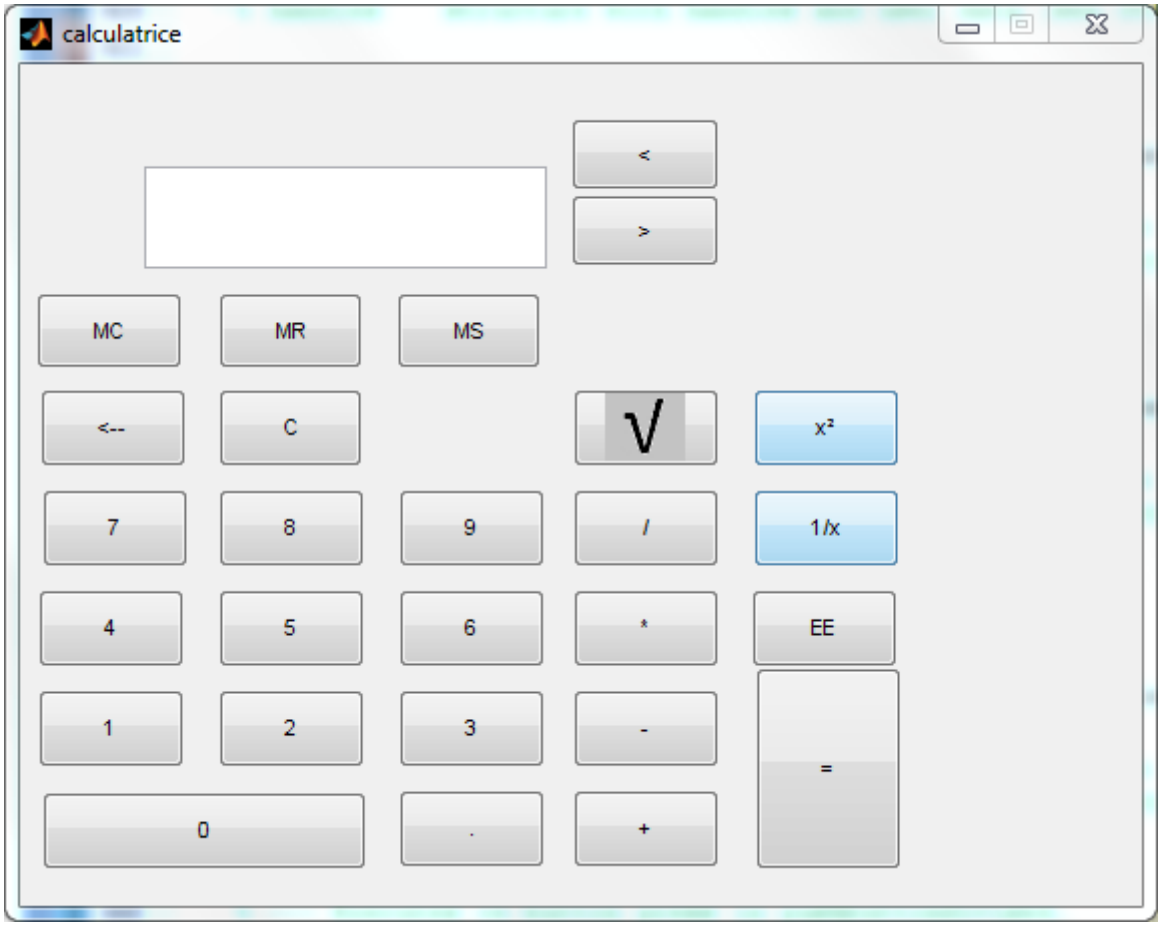


Qui permet de choisir dans répertoire que vous allez mettre votre projet et le nom du point fig par exemple calculatrice.

Faite ensuite OK.

Étape 1:Création de l'aspect visuel de l'interface

Placer les éléments suivantes :



Puis compiler votre interface pour générer le code.

Indice: *Pour faire apparaître la racine carré:*

Dans le code:

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


handles.output = hObject;




functiondir=getFunctionFolder();%Given choosing
handles.path = [functiondir '\picture\'];
```

Puis la fonction que je vous donne:

```
function functiondir=getFunctionFolder()
functionname='calculatrice.m';
functiondir=which(functionname);
```

Étape 2: Gestion des chiffres de 0 à 9 et de leurs affichages

1. Nous devons pour un premier temps créer la fonction " append_num(handles, num)", prends comme paramètres *handles* le constructeur de l'affichage, *num* est le chiffre que nous voulons faire afficher.

L'algorithme que je propose de codé est le suivant:

Lecture des opérateurs '+', '-', '*', '/'

Si vide (lecture des opérateurs)

Alors on affiche l'expression des opérateurs concaténer aux chiffres ou au nombre

Sinon Si la comparaison des opérateurs sont différents de 0

Alors on affiche l'expression des opérateur concaténer par un espace puis un nombre ou des chiffres

2. Nous devons ensuite appeler cette fonction sur toute les chiffres de 0 à 9 et le point (qui est le point du décimale).

Étape 3: Les opérateurs de calculs et de leurs affichages

1. Nous devons pour un premier temps créer la fonction " append_operator(handles, operator)", prends comme paramètres *handles* le constructeur de l'affichage, *operator* sont les opérateurs '+', '/', '#', '-'.

L'algorithme que je propose de codé est le suivant:

Lecture des éléments de l'affichage

Concaténer les nombres et l'opérateur pour afficher

2. Nous devons ensuite appeler cette fonction sur toute les opérations '+', '/', '#', '-'.
3. Construction de la deuxième fonction qui va gérer les opérations spécifiques qui sont : sqrt pour la racine carré, x^2, 1/x et EE.

Nous devrons pour ces opérateurs suivre la syntaxe suivante:

Pour simplifier notre calculatrice, on ne peut pas multiplier un chiffre avant l'opérateur.
sqrt : l'affichage doit-être comme suivant *sqrt(valeur)*
x^2 : l'affichage doit-être comme suivant *valeur ^2*
1/x : l'affichage doit-être comme suivant *1/valeur*
EE : l'affichage doit-être comme suivant 10^valeur

Je l'appellerai "*apprend_operator2(handles, operator)*".

Lecture des éléments de l'affichage

Concaténer les nombres et l'opérateur pour afficher

Indices :

- utiliser set, get, strcmp et any
- faites des conditions pour chaque opérateurs

4. Faites ensuite le rappel cette fonction ci-dessous écrite à chaque bouton opération.

Étape 4: La gestion du bouton égale et de l'affichage du résultat

Je vais appeler cette fonction "*pushbuttonEgal_Callback(hObject, evendata, handles)*".

Je vous donne le code car un peu complexe à écrire.

```
% --- Executes on button press in pushbuttonEgal.
function pushbuttonEgal_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonEgal (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

%retrieve expression in edit box
userExpression = get(handles.edit_calc_display,'string');

% perform action if something has been entered
if ~isempty(userExpression)
    try
        % evaluate character array that was entered by user and store as a
        % number
        answerNum = eval(userExpression);

        % convert the number to a string
        answerStr = num2str(answerNum);

        % overwrite user's input to display that answer
        set(handles.edit_calc_display,'string', answerStr)

        %update the history of expressions that were entered so that we can
        %go back if we want using the up and down arrows
        update_history(handles, userExpression)
    catch
        %the expression was not valid, so notify the user by c ng text
        %in the edit box
        set(handles.edit_calc_display,'string', 'Syntax Error', ...
            'ForegroundColor',[0.6,0.6,0.6], ...
            'FontAngle','Italic', ...
            'FontWeight','Normal')

        %display the message for 1.5 seconds
        pause(1.5)

        %and change it back to what was originally entered
        set(handles.edit_calc_display,'string',userExpression, ...
```

1 2 3 4 5 6 7

```
            'ForegroundColor',[0,0,0], ...
            'FontAngle','Normal', ...
            'FontWeight','Bold')
    end
end
```

1. Faites l'explication des ronds numérotés du code ci-dessus:

1

2

3

4

5

6

7

2. La fonction update_history permet de faire l'historique des opérations effectués précédents:

Recopier ce code:

```
function  update_history(handles, userExpression)
%insert new expression into hisotrical list (a cell array) of expressions
curHistoryIndex = handles.UserData.historyIndex;
handles.UserData.history =
vertcat(handles.UserData.history(1:curHistoryIndex), {userExpression});

%increment curent history index
handles.UserData.historyIndex = curHistoryIndex+1;

% Update handles structure
guidata(handles.edit_calc_display, handles);
```

3. Recopier ce code et le comprendre :

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

%récupération des éléments afficher
userExpression = get(handles.edit_calc_display,'string');
```

```
%get current place in line of history
curHistoryIndex = handles.UserData.historyIndex;

% si l'historique est vide
if curHistoryIndex > 0

    %construct what the new expression should be
    newExpression = handles.UserData.history{curHistoryIndex};
    if ~strcmp(userExpression,newExpression)
        %it doesnt match whats currently in the box, so use it
        set(handles.edit_calc_display,'string', newExpression)
        if curHistoryIndex > 1
            % change the index to reflect what is now being displayed
            handles.UserData.historyIndex =  handles.UserData.historyIndex
- 1;
        end
    elseif curHistoryIndex-1 > 0
        %it DOES match what's in the box, so the user probably clicked NEXT
        %and then clicked PREV in which case we need to change two spaces
        %in our history instead of one

        %construct a different new expression
        newExpression = handles.UserData.history{curHistoryIndex-1};

        if ~strcmp(userExpression,newExpression)
            %it's different from what's in the box, so let's use it
            set(handles.edit_calc_display,'string', newExpression)

            if curHistoryIndex > 2
                % change the index to reflect what is now being displayed
                handles.UserData.historyIndex =
handles.UserData.historyIndex - 2;
            end
        end
    end
```

```
end

%update handles structure to ensure changes we made will be stored for
%later
guidata(handles.edit_calc_display, handles)
```

Indice: Je déclare dans la fonction d'ouverture de l'application calculatruce_OpeningFcn

```
%initialize history
handles.UserData.history = {};
handles.UserData.historyIndex = 0;
```

4. Faite le même processus pour la fonction : `pushbuttonSuivant_Callback(hObject, eventdata, handles)`

Étape 5: Les touches MR, MC, MS:

MR est le rappel mémoire.

MC permet de vider la mémoire.

MS active la mémoire

Les 3 codes sont les suivants:

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

```
handles.UserData.stored = '';
set(handles.text_memory,'string','')

% Update handles structure so that it is preserved during future callbacks
guidata(hObject, handles);



% --- Executes on button press in pushbuttonMR.
function pushbuttonMR_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonMR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%change the text in the window to whatever was stored
if ~isempty(handles.UserData.stored)
    set(handles.edit_calc_display,'string',handles.UserData.stored)
end

 % --- Executes on button press in pushbuttonMS.
function pushbuttonMS_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonMS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% store the current string to a variable in the handles structure
handles.UserData.stored = get(handles.edit_calc_display,'string');
if ~isempty(handles.UserData.stored)
    set(handles.text_memory,'string','M')
end

% Update handles structure so that it is preserved during future callbacks
guidata(hObject, handles);
```

Étape 6: La touche Clear:

Remise à zéro l'affichage

Étape 7: Effacer un par un les chiffres

Voici un petit algorithme :

Lire les éléments de l'affichage

Si il y a quelques choses dans l'affichage

Si comparaison entre caractère vide et que le nombre de caractère est supérieur à 1

Enlever les deux dernier caractères

Sinon

Étape 8: Gestion des touches claviers

1. Je souhaite faire égale dans l'affichage après avoir fait par exemple cette opération 2+5 puis entrer. Vous devez créer la méthode KeyPressFcn dans la zone de l'affichage du calcul.

Indice: Utiliser un condition que je peux écrire comme :

```
if strcmp(get(gcf,'CurrentKey'),'return')
  Votre code

end
```

et l'utilisation du remise à jour de l'écran par drawnow.

2. Je termine mon application par ces deux lignes:

```
% --- Executes on key press with focus on figure1 and none of its controls.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
```

```
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)


if strcmp(get(gcf,'CurrentKey'),'return')
    pushbuttonEgal_Callback(hObject, eventdata, handles)
end
```

Correction :

```
function varargout = calculatrice(varargin)
% CALCULATRICE M-file for calculatrice.fig
%      CALCULATRICE, by itself, creates a new CALCULATRICE or raises the
existing
%      singleton*.
%
%      H = CALCULATRICE returns the handle to a new CALCULATRICE or the
handle to
%      the existing singleton*.
%
%      CALCULATRICE('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in CALCULATRICE.M with the given input
arguments.
%
%      CALCULATRICE('Property','Value',...) creates a new CALCULATRICE or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before calculatrice_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to calculatrice_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help calculatrice

% Last Modified by GUIDE v2.5 11-Nov-2013 16:15:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @calculatrice_OpeningFcn, ...
                   'gui_OutputFcn',  @calculatrice_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 calculatrice is made visible.
function calculatrice_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 calculatrice (see VARARGIN)

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

%initialize history
handles.UserData.history = {};
handles.UserData.historyIndex = 0;


functiondir=getFunctionFolder();%Given choosing
handles.path = [functiondir '\picture\'];

set(handles.pushbuttonracine, 'CData', imread([handles.path '\'
'racine.jpg']));



% Update handles structure
guidata(hObject, handles);

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



%% Fonction de gestion des répertoiers annexes
function functiondir=getFunctionFolder()
functionname='calculatrice.m';
functiondir=which(functionname);
functiondir=functiondir(1:end-length(functionname));




% --- Outputs from this function are returned to the command line.
function varargout = calculatrice_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;



function edit_calc_display_Callback(hObject, eventdata, handles)
% hObject    handle to edit_calc_display (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,'String') returns contents of edit_calc_display as
text
```

```
%        str2double(get(hObject,'String')) returns contents of
edit_calc_display as a double


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

% Hint: edit 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


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

%récupération des éléments afficher
userExpression = get(handles.edit_calc_display,'string');

%get current place in line of history
curHistoryIndex = handles.UserData.historyIndex;

% si l'historique est vide
if curHistoryIndex > 0

    %construct what the new expression should be
    newExpression = handles.UserData.history{curHistoryIndex};
    if ~strcmp(userExpression,newExpression)
        %it doesnt match whats currently in the box, so use it
        set(handles.edit_calc_display,'string', newExpression)
        if curHistoryIndex > 1
            % change the index to reflect what is now being displayed
            handles.UserData.historyIndex =  handles.UserData.historyIndex
- 1;
        end
    elseif curHistoryIndex-1 > 0
        %it DOES match what's in the box, so the user probably clicked NEXT
        %and then clicked PREV in which case we need to change two spaces
        %in our history instead of one

        %construct a different new expression
        newExpression = handles.UserData.history{curHistoryIndex-1};

        if ~strcmp(userExpression,newExpression)
            %it's different from what's in the box, so let's use it
            set(handles.edit_calc_display,'string', newExpression)

            if curHistoryIndex > 2
```

```
                % change the index to reflect what is now being displayed
                handles.UserData.historyIndex =
handles.UserData.historyIndex - 2;
            end
        end
    end
end

%update handles structure to ensure changes we made will be stored for
%later
guidata(handles.edit_calc_display, handles)




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

%get current expression
userExpression = get(handles.edit_calc_display,'string');

%get current place in line of history
curHistoryIndex = handles.UserData.historyIndex;

%check to make sure we can go forward in the history list from where the
%current placeholder is
if size(handles.UserData.history,1) > curHistoryIndex

    %construct what the new expression should be
    newExpression = handles.UserData.history{curHistoryIndex+1};

    if ~strcmp(userExpression, newExpression)
        %it doesnt match whats currently in the box, so use it
        set(handles.edit_calc_display,'string',newExpression)
        % change the index to reflect what is now being displayed
        handles.UserData.historyIndex = handles.UserData.historyIndex +1;
        guidata(handles.edit_calc_display, handles);
    elseif size(handles.UserData.history,1) > curHistoryIndex+1
        %it DOES match what's in the box, so the user probably clicked PREV
        %and then clicked NEXT in which case we need to change two spaces
        %in our history instead of one

        %construct a different new expression
        newExpression = handles.UserData.history{curHistoryIndex+2};
        set(handles.edit_calc_display,'string',newExpression)

        % change the index to reflect what is now being displayed
        handles.UserData.historyIndex = handles.UserData.historyIndex +2;
    end

elseif size(handles.UserData.history,1) == curHistoryIndex &&
curHistoryIndex>0
    % we are at the end of the road, i.e. latest in history chain. Simply
```

```
    % update display with the last entry and leave index where it is.

set(handles.edit_calc_display,'string',handles.UserData.history{curHistoryI
ndex})
end

%update handles structure to ensure changes we made will be stored for
%later
guidata(handles.edit_calc_display, handles)




% --- Executes on button press in pushbutton0.
function pushbutton0_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton0 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,0);



% --- Executes on button press in pushbutton3.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,1);

% --- Executes on button press in pushbutton3.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,2);


% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,3);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,4);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,5);

% --- Executes on button press in pushbutton6.
```

```
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,6);

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,7);



% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,8);

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_num(handles,9);

function apprend_num(handles, num)

% Gestion de l'affichage
userExpression = get(handles.edit_calc_display, 'string');


if isempty(userExpression)

    set(handles.edit_calc_display,'string',[userExpression num2str(num)]);

else
    if any(strcmp(userExpression(end),{'+','-','*','/'}))
        % if so, add a space between the operator and the number
        set(handles.edit_calc_display,'string',[userExpression ' '
num2str(num)]);
    else
        %otherwise just insert
        set(handles.edit_calc_display,'string',[userExpression
num2str(num)]);
    end


end
```

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

handles.UserData.stored = '';
set(handles.text_memory,'string','')

% Update handles structure so that it is preserved during future callbacks
guidata(hObject, handles);



% --- Executes on button press in pushbuttonMR.
function pushbuttonMR_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonMR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%change the text in the window to whatever was stored
if ~isempty(handles.UserData.stored)
    set(handles.edit_calc_display,'string',handles.UserData.stored)
end




% --- Executes on button press in pushbuttonMS.
function pushbuttonMS_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonMS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% store the current string to a variable in the handles structure
handles.UserData.stored = get(handles.edit_calc_display,'string');
if ~isempty(handles.UserData.stored)
    set(handles.text_memory,'string','M')
end

% Update handles structure so that it is preserved during future callbacks
guidata(hObject, handles);


% --- Executes on button press in pushbuttonFleche.
function pushbuttonFleche_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonFleche (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
userExpression = get(handles.edit_calc_display,'string');
```

```
if ~isempty(userExpression)
    if strcmp(userExpression(end), ' ') && length(userExpression) > 1
        %remove the trailing space and whatever comes before it
        set(handles.edit_calc_display,'string',userExpression(1:end-2))
    else
        %simply remove the last character
        set(handles.edit_calc_display,'string',userExpression(1:end-1))
    end
end

% --- Executes on button press in pushbuttonC.
function pushbuttonC_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%set the text to blank
set(handles.edit_calc_display,'string','')

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (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 pushbuttonracine.
function pushbuttonracine_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonracine (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_operator2(handles, 'sqrt');

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_operator2(handles, '1/');


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

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

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

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

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

% --- Executes on button press in pushbuttonExposant.
function pushbuttonExposant_Callback(hObject, eventdata, handles)
% hObject    handle to pushbuttonExposant (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
apprend_operator2(handles, 'x');


% --- Executes on button press in pushbutton28.
function pushbutton28_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton28 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

apprend_operator2(handles, '10^');




function apprend_operator(handles, operator)


%get the current expression
userExpression = get(handles.edit_calc_display,'string');

%and concatenate the operator
set(handles.edit_calc_display,'string',[userExpression ' ' operator ' ']);


function apprend_operator2(handles, operator)


%get the current expression
userExpression = get(handles.edit_calc_display,'string');


if any(strcmp(operator,{'sqrt'}))
%and concatenate the operator

set(handles.edit_calc_display,'string',[  operator '(' userExpression ')
']);
```

```
else if  any(strcmp(operator(end),{'x'}))
        set(handles.edit_calc_display,'string',[userExpression '^2']);
    else
       if  any(strcmp(operator,{'1/'}))
           set(handles.edit_calc_display,'string',[operator userExpression
]);
       else
           if any(strcmp(operator, {'10^'}))

            set(handles.edit_calc_display,'string',[ userExpression ''
operator  userExpression ]);
           end

        end
    end

end

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

%retrieve expression in edit box
userExpression = get(handles.edit_calc_display,'string');

% perform action if something has been entered
if ~isempty(userExpression)
    try
        % evaluate character array that was entered by user and store as a
        % number
        answerNum = eval(userExpression);

        % convert the number to a string
        answerStr = num2str(answerNum);

        % overwrite user's input to display that answer
        set(handles.edit_calc_display,'string', answerStr)

        %update the history of expressions that were entered so that we can
        %go back if we want using the up and down arrows
        update_history(handles, userExpression)
    catch
        %the expression was not valid, so notify the user by changing text
        %in the edit box
        set(handles.edit_calc_display,'string', 'Syntax Error', ...
            'ForegroundColor',[0.6,0.6,0.6], ...
            'FontAngle','Italic', ...
            'FontWeight','Normal')

        %display the message for 1.5 seconds
        pause(1.5)

        %and change it back to what was originally entered
        set(handles.edit_calc_display,'string',userExpression, ...
            'ForegroundColor',[0,0,0], ...
            'FontAngle','Normal', ...
            'FontWeight','Bold')
```

```
    end
end


function  update_history(handles, userExpression)
%insert new expression into hisotrical list (a cell array) of expressions
curHistoryIndex = handles.UserData.historyIndex;
handles.UserData.history =
vertcat(handles.UserData.history(1:curHistoryIndex), {userExpression});

%increment curent history index
handles.UserData.historyIndex = curHistoryIndex+1;

% Update handles structure
guidata(handles.edit_calc_display, handles);



function text_memory_Callback(hObject, eventdata, handles)
% hObject    handle to text_memory (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,'String') returns contents of text_memory as text
%        str2double(get(hObject,'String')) returns contents of text_memory
as a double


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

% Hint: edit 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


% --- Executes on key press with focus on edit_calc_display and none of its
controls.
function edit_calc_display_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to edit_calc_display (see GCBO)
% eventdata  structure with the following fields (see UICONTROL)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)



% update answer (need to call function twice due to bug with keypressfcn
if strcmp(get(gcf,'CurrentKey'),'return')
    pushbuttonEgal_Callback(hObject, eventdata, handles)
    pause(0.01)
```

```
    drawnow
    pushbuttonEgal_Callback(hObject, eventdata, handles)
    pause(0.01)
    drawnow
end


% --- Executes on key press with focus on figure1 and none of its controls.
function figure1_KeyPressFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  structure with the following fields (see FIGURE)
%   Key: name of the key that was pressed, in lower case
%   Character: character interpretation of the key(s) that was pressed
%   Modifier: name(s) of the modifier key(s) (i.e., control, shift) pressed
% handles    structure with handles and user data (see GUIDATA)


if strcmp(get(gcf,'CurrentKey'),'return')
    pushbuttonEgal_Callback(hObject, eventdata, handles)
end
```