Chapitre 7

Programmation avec MATLAB

I. Opérateurs arithmétiques, logiques et caractères spéciaux

I.1. Opérateurs et caractères spéciaux

Le tableau suivant résume les opérateurs arithmétiques, logiques et spéciaux de MATLAB.

I.1.1. Opérateurs arithmétiques

symbole	*fonction*
+	addition de réels et de matrices
−	soustraction de réels et de matrices
*	produit de réels et de matrices
.*	produit élément par élément de matrices
^	élévation à une puissance de réels et de matrices
.^	puissance élément par élément de matrices

N. Martaj, M. Mokhtari, *MATLAB R2009, SIMULINK et STATEFLOW pour Ingénieurs, Chercheurs et Etudiants*, DOI 10.1007/978-3-642-11764-0_7, © Springer-Verlag Berlin Heidelberg 2010

\	division à gauche de réels et de matrices
/	division à droite de réels et de matrices (division classique)
./	division élément par élément de matrices

I.1.2. Opérateurs relationnels

==	égalité
~=	différent
<	strictement inférieur
<=	inférieur ou égal
>	strictement supérieur
>=	supérieur ou égal
&	ET logique (AND)
|	OU logique (OR)
~	NON logique (NOT)
xor	OU exclusif (XOR)

Des commandes telles que `and`, `or`, … réalisent les mêmes fonctions que & et |.

```
>> a=1; b=0; a&b
ans =
     0
>> a=1; b=0; a|b
ans =
     1
>> or(a,b)
ans =
     1
>> and(a,b)
ans =
     0
```

I.1.3. Caractères spéciaux

Nous donnons ci-après, les caractères spéciaux uniquement propres à MATLAB.

%	commentaires
'	transposée de matrices et délimiteur de chaînes de caractères
!	commandes système (échappement)
,	séparation d'instructions ou de réels dans une matrice ou de commandes dans une même ligne.
;	séparation d'instructions (pas d'affichage de valeurs intermédiaires) ou des lignes d'une matrice
...	symbole de continuation (pour terminer une instruction ou une expression à la ligne suivante)
..	répertoire parent du répertoire en cours
[ ]	définition de vecteurs ou de matrices

Dans les lignes de commandes suivantes, nous allons utiliser certains de ces caractères.

fichier carac_spec.m

```
% matrice carrée
A = [1 3;4 6] ;
At = A'  % La transposée de A

% Produit de At par A et inverse

AtA=At*A, inv_AtA=inv(AtA)
A =
     1     3
     4     6

At =
     1     4
     3     6


AtA =
    17    27
    27    45

inv_AtA =

    1.2500   -0.7500
   -0.7500    0.4722
```

I.2. Fonctions retournant une valeur logique

MATLAB fournit un certain nombre de fonctions qui retournent des valeurs logiques (vrai : >=1 ou faux : 0) pour tester l'existence de variables, leur type, etc.
Nous en présentons ci-dessous les plus importantes.

- **exist**

```
val = exist('var')
```

Permet de tester si la variable, la fonction ou le fichier `Var` est défini. Elle retourne :

`0` si `Var` n'existe pas (ni fonction, ni variable),
`1` si `Var` est une variable de l'espace de travail,
`2` si `Var` est un fichier M qui se trouve dans un des répertoires des chemins de recherche de MATLAB,
`3` si `Var` est un fichier MEX qui se trouve dans un des répertoires des chemins de recherche de MATLAB,
`4` si `Var` est un modèle SIMULINK (fichier `.mdl`),
`5` si `Var` est une fonction du noyau de MATLAB (`built-in function`),
`6` si Var est un fichier P trouvé dans les chemins de recherché de MATLAB,
`7` si Var est un répertoire.

Il existe d'autres options de spécification de la nature de la variable qu'on recherche.

```
val = exist('x','var')
```

ne recherche que des variables.

```
val = exist('x','file')
```

ne recherche que des variables `x` en tant que répertoires.

```
>>  clear all
>>  exist('x')
ans =
     0
```

```
>>  x = randn(2)
x =
    0.5377   -2.2588
    1.8339    0.8622
```

```
>>  exist('x')
ans =
     1
```

```
>> exist('sol_syst_sur_determine')  % modèle SIMULINK
ans =
     4
```

La fonction `inv` (inversion de matrices) est présente dans le noyau de MATLAB.

```
>>  exist('inv') % built-in function
ans =
     5
```

```
>> exist('sinc2')  % fichier M
ans =
     2
```

- **all**

```
val = all(Var)
```

Retourne la valeur "vrai" si tous les éléments du vecteur `Var` sont vrais (différents de `0`). Si `Var` est une matrice, l'opération est réalisée, par défaut, sur chaque colonne et la fonction `all` retourne un vecteur de type ligne, constitué de `0` et `1`.

```
>>  all(x)
ans =
     0     0     1
```

Seule la 3ème colonne possède tous ses éléments non nuls.

```
>>  all(all(x>=0))
ans =
     1
```

L'opération peut être réalisée sur chaque colonne, et dans ce cas le résultat sera un vecteur colonne, si on spécifié la dimension 2 avec la syntaxe :

```
val = all(Var, dim)
```

```
>> all(x,2)
ans =
     0
     0
```

Il est inutile de spécifier la dimension 1 dans le cas d'une matrice 2D.

Cette propriété est intéressante dans le cas d'un tableau multidimensionnel pour lequel on peut considérer la dimension supérieure ou égale à 3 (On se référera au chapitre des tableaux multidimensionnels)

```
>> y=cat(3,x, eye(3), randn(3))

y(:,:,1) =
     1     0     3
     0     0     1
     5     7     0

y(:,:,2) =
     1     0     0
     0     1     0
     0     0     1

y(:,:,3) =

    1.4090   -1.2075    0.4889
    1.4172    0.7172    1.0347
    0.6715    1.6302    0.7269
```

```
>> all(y,3)
ans =
     1     0     0
     0     0     0
     0     0     0
```

Pour tester tous les éléments du tableau `y`, nous devons imbriquer 3 commandes `all`.

```
>>  all(all(all(y)))
ans =
     0
```

- **any**

```
val = any(Var)
```

Retourne la valeur "vrai" si au moins un élément du vecteur `Var` est vrai (différent de `0`). Si `Var` est une matrice, l'opération est réalisée sur chaque vecteur colonne et la valeur retournée est un vecteur ligne, constitué de `0` et `1`.

```
>> x = [1 0 3; 0 0 1]

x =
     1     0     3
     0     0     1

>> any(x)
ans =
     1     0     1
```

Pareil que pour la commande `all`, on peut spécifier la dimension 2, supérieure ou égale à 3 pour un tableau multidimensionnel.

```
>> any(x,2)
ans =
     1
     1

>> any(y,3)
ans =
     1     1     1
     1     1     1
     1     1     1

>> any(any(y,3))
ans =
     1     1     1
```

- **find**

```
I = find(Exp), [I,J] = find(Exp)
```

Les arguments de retour sont les indices des éléments non nuls du tableau `Exp`.

`Exp` : matrice ou expression logique sur des matrices,
`I` : n° des éléments, la numérotation se fait ligne par ligne et de gauche à droite,

`[I, J]` : les vecteurs `I` et `J` représentent respectivement les indices des lignes et des colonnes.

Les éléments non nuls de la matrice `x` définie précédemment n'ont pour numéros 1 et 6 comme le montre l'instruction suivante.

```
>>  i = find(x==1)
ans =
     1
     6
```

Seuls les éléments `x(1,1)` et `x(2,3)` sont égaux à 1.

```
>>  [i j] = find(x==1)
i =
     1
     2
j =
     1
     3
```

- **isnan**

```
val = isnan(Var)
```

La valeur de retour `val` est une matrice de mêmes dimensions que `Var`. L'élément `val(i,j)` est `0` si `Var(i,j)` est un nombre fini ou infini `(Inf)` et `1` dans le cas où `Var(i,j)` correspond à `NaN`.

```
>> x=[1 0/0; inf/0 5]
x =
     1   NaN
   Inf     5
```

```
>> isnan(x)
ans =
     0     1
     0     0
```

- **isinf**

```
val = isinf(Var)
```

La valeur de retour `val` est une matrice de mêmes dimensions que `Var`. L'élément `val(i,j)` est `1` si `Var(i,j)` est un nombre infini `(Inf)` et `0` dans le cas où `Var(i,j)` correspond à `NaN` ou à un nombre fini.

```
>>  isinf(x)
ans =

     0     0
     1     0

>> isinf(x)|isnan(x)
ans =
     0     1
     1     0
```

- **isfinite**

```
val = isfinite(Var)
```

La valeur de retour `val` est une matrice de mêmes dimensions que `Var`. L'élément `val(i,j)` est `1` si `Var(i,j)` est un nombre fini et `0` dans le cas où `Var(i,j)` correspond à `NaN` ou à nombre infini `(Inf)`.

```
>> x=[1 0/0; inf/0 5]
>> isfinite(x)
ans =
     1     0
     0     1
```

- **isempty**

```
val = isempty(Var)
```

`val` prend la valeur `1` si `Var` est une matrice vide (0 ligne ou 0 colonne) et `0` dans le cas contraire. Une matrice vide est symbolisée par `[ ]`.

```
>> isempty(x)
ans =
     0

>>  x = [];
>>  isempty(x)
ans =
     1
```

- **isreal**

```
val = isreal(Var)
```

`val` prend la valeur `1` si `Var` est une matrice ne contenant que des nombres réels et `0` dans le cas où `Var` contient au moins un nombre complexe.

```
>>  isreal(x)
ans =
     1
```

```
>>  clear all
>>  w = [1+i 2; 1-i -5];
>>  isreal(w)
ans =
     0
```

- **issparse**

```
val = issparse(Mat)
```

Retourne `1` si la matrice `Mat` est une matrice creuse et `0` dans le cas contraire.

```
>> x=sprand(2)
x =
   (1,1)        0.6324

>> issparse(x)
ans =
     1
```

- **isstr**

Retourne `1` si `Var` est une chaîne de caractères et `0` dans les cas contraires.

```
val = isstr(Var)
```

```
>> isstr('MATLAB')
ans =
     1
```

- **isglobal**

```
val = isglobal(Var)
```

Retourne `1` si la variable `Var` est définie comme globale (entre l'espace de travail MATLAB et une fonction) ou seulement locale à la fonction.

II. Evaluation de commandes en chaînes de caractères

Des commandes peuvent être insérées dans une chaîne de caractères et évaluées ensuite par d'autres commandes.

- **eval**

```
eval (chaîne)
```

Evalue une chaîne de caractères en exécutant les instructions qui la composent.

Pour récupérer les arguments de sortie de l'expression `chaîne`, on utilisera la forme suivante de la commande.

```
[x,y,z, ...] = eval (chaîne)
```

```
>> clear all
>> x=[1 2 ;4 7] ;
>> eval('inv(x)')

ans =
    -7     2
     4    -1
```

Exemple

On peut concaténer plusieurs commandes et les évaluer comme dans l'exemple suivant de la génération de 5 matrices carrées de nombres aléatoires, d'ordre 3, nommées M1 à M5.

```
for i = 1:5
 eval(['M' num2str(i) ' = rand(3);']);
end
```

```
>>  whos

  Name        Size              Bytes  Class     Attributes
  A           2x2                  32  double
  M1          3x3                  72  double
  M2          3x3                  72  double
  M3          3x3                  72  double
  M4          3x3                  72  double
  M5          3x3                  72  double
  ans         2x2                  32  double
  i           1x1                   8  double
  x           2x2                  32  double
```

```
>>  M1
M1 =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

- **feval**

Cette commande évalue la fonction `fonct` avec ses paramètres d'appel `x1, ..., xn`.

```
feval(fonct, x1, x2, ... , xn)
```

`fonct :` fonction définie dans un fichier M (`fonct.m`),
`x1, x2, ... , xn :` arguments d'appel de la fonction.

Exemple

fichier sommep.m

```
function a = somme(x)
% somme des éléments de la matrice x
a = sum(x);
```

- Appel direct de la fonction :

```
>> a=[1 2 ;4 7] ;
>> som=somme(a)
som=
 14
```

- Appel en utilisant la fonction `feval`:

```
>>  som2 = feval('somme',a)
som2 =
    14
```

- **evalc**

La commande `[T, A]=evalc(chaine)` est équivalente à `A = eval(chaine)`, sauf que `T` reçoit les éventuels messages d'erreur.

```
>> [T, A]=evalc('det(x)')
T =
     ''
A =
    -1
```

Dans ce cas, le déterminant vaut -1 sans aucun message d'erreur.

- **evalin**

La commande `evalin` permet d'avoir le même résultat comme suit :

```
>> evalin('base','som3= somme(x)')
som3 =
    14
```

- **lasterr**

Retourne le dernier message d'erreur généré par MATLAB.

```
>>  x = 1/0;
Warning: Divide by zero
```

```
>>  xx
??? Undefined function or variable xx.
>>  lasterr
ans =
Undefined function or variable xx.
```

- **Inline**

Il est parfois très utile de définir une fonction qui sera employée pendant la session courante de MATLAB seulement. MATLAB possède une commande en ligne employée pour définir les fonctions, dites `inline` ou « en ligne >>.

```
>> g = inline('sin(2*pi*f + theta)')
g =
     Inline function:
     g(f,theta) = sin(2*pi*f + theta)
```

```
f = inline('inv(transpose(A)*A)*transpose(A)*B','A','B')

A=[1, 2; 4 5];
B=[5 6]';

f(A,B)
ans =

   -4.3333
    4.6667
```

La commande `char` transforme le corps de la fonction en chaîne de caractères.

```
>> char(f)

ans =
inv(transpose(A)*A)*transpose(A)*B
```

Les arguments sont récupérés par la commande `argnames`.

```
>> argnames(f)
ans =
    'A'
    'B'
```

III. Commandes structurées

III.1. Boucle for

La boucle `for` possède la syntaxe suivante :

```
for k = val_init : pas : val_fin
liste des instructions
end
```

Le gros avantage de MATLAB est la vectorisation des données, ce qui permet très souvent d'éviter l'utilisation de la boucle `for`.

Considérons l'exemple suivant que l'on peut réaliser avec et sans boucle `for`. Nous calculerons le temps de calcul et estimerons le temps que l'on gagne si l'on évite la boucle `for`.

Nous disposons de 2 vecteurs `x` et `y`. La méthode d'interpolation de Cramer qui permet un régression linéaire `y(x)` par la droite `y = a x + b` donne les formules suivantes pour le calcul des coefficients `a` et `b`.

Nous considérons le cas de l'interpolation de courbes réelles dont le calcul des paramètres nécessite l'utilisation de calculs matriciels.

$$a=\frac{\det\begin{bmatrix}\sum_{k=1}^{N} z(k) & \sum_{k=1}^{N} x(k)\\ \sum_{k=1}^{N} x(k)\ z(k) & \sum_{k=1}^{N} x(k)^2\end{bmatrix}}{\det\begin{bmatrix}N & \sum_{k=1}^{N} x(k)\\ \sum_{k=1}^{N} x(k) & \sum_{k=1}^{N} x(k)^2\end{bmatrix}},\quad b=\frac{\det\begin{bmatrix}N & \sum_{k=1}^{N} z(k)\\ \sum_{k=1}^{N} x(k) & \sum_{k=1}^{N} x(k)z(k)\end{bmatrix}}{\det\begin{bmatrix}N & \sum_{k=1}^{N} x(k)\\ \sum_{k=1}^{N} x(k) & \sum_{k=1}^{N} x(k)^2\end{bmatrix}}$$

Considérons uniquement le numérateur du coefficient `a`.

$$num\_a=\det\begin{bmatrix}\sum_{k=1}^{N} z(k) & \sum_{k=1}^{N} x(k)\\ \sum_{k=1}^{N} x(k)\ z(k) & \sum_{k=1}^{N} x(k)^2\end{bmatrix},$$

Le calcul, en utilisant la boucle `for` donne le script suivant.

fichier calcul_avec_for.m

```
x = [0.10 0.20 0.50 1.0  1.50 1.90 2.00 3.00 4.00 6.00]
y = [0.95 0.89 0.79 0.70 0.63 0.58 0.56 0.45 0.36 0.28]

clear all, close all, format long, clc
x = [0.1 0.2 0.5 1.0 1.5 1.9 2.0 3.0 4.0 6.0];
y = [0.95 0.89 0.79 0.70 0.63 0.58 0.56 0.45 0.36 0.28];
% Initialisation des variables
num_a11=0;
num_a12=0;
num_a21=0;
num_a22=0;
for k= 1 : length(x)
    num_a11 = num_a11+y(k);
    num_a12 = num_a12+x(k);
    num_a21=num_a21+x(k)*y(k);
    num_a22=num_a22+x(k)^2;
end
num_a=det([num_a11 num_a12; num_a21 num_a22])
```

Nous pouvons aussi remplir directement la matrice directement à chaque pas de la boucle.

```
mat_num_a=0 ;
for k= 1 : length(x)
    mat_num_a= mat_num_a+[y(k) x(k);x(k)*y(k) x(k)^2];
end
```

MATLAB redimensionne les variables, `mat_num_a` est initialisée à 0 comme une variable scalaire mais elle est remplie comme une matrice 2D à l'intérieur de la boucle.

Nous trouvons le résultat suivant :

```
num_a =
    2.647694000000001e+002
```

En utilisant la propriété de vectorisation et les fonctions de calcul propres aux vecteurs et matrices, on peut utiliser le script simplifié suivant :

fichier calcul_sans_for.m

```
clear all, close all, format long, clc
x = [0.1 0.2 0.5 1.0 1.5 1.9 2.0 3.0 4.0 6.0];
y = [0.95 0.89 0.79 0.70 0.63 0.58 0.56 0.45 0.36 0.28];
det_mat_num=det([sum(y) sum(x);sum(x.*y) sum(x.^2)])
```

Nous obtenons la même valeur :

```
det_mat_num =
    2.647694000000001e+002
```

Nous allons maintenant comparer les temps de calcul que nous déterminons en utilisant les commandes `tic` et `toc` .

fichier temps_avec_for.m

```
tic
calcul_avec_for %  appel du script calcul_avec_for.m
toc
```

et

fichier temps_sans_for.m

```
tic
calcul_sans_for % appel du script calcul_sans_for.m
toc
```

Nous trouvons les résultats suivants pour les temps de calcul.

```
Avec boucle for :
det_mat_num =
    2.647694000000001e+002
Elapsed time is 0.001019 seconds.

Sans boucle for :
det_mat_num =
    2.647694000000001e+002
Elapsed time is 0.000404 seconds.
```

Le temps de calcul est 2,6 fois supérieur pour le calcul avec boucles que sans. Ce rapport augmente avec la taille des vecteurs `x` et `y`.
Néanmoins, ce rapport sera faible avec la puissance sans cesse croissante des processeurs.

Dans de nombreux cas, la boucle `for` peut être remplacée avantageusement par des opérations matricielles. L'exemple suivant en est une bonne illustration.

On désire calculer les valeurs du vecteur `x` de dimension `n` dont les composantes sont définies par l'expression suivante :

$$x(i)=\sum_{j=1}^{m} \exp(i)\,\log(j^2) \quad i=1,2,\ldots,n$$

Solution avec les boucles for (fichier fic_for1.m)

```
function x = fic_for1(n,m)
for i = 1:n
    xtemp = 0;
  for j = 1:m
    xtemp= xtemp + exp(i)*log(j^2);
  end
    x(i)=xtemp;
end
```

```
>>  fic_for1(4,5)
ans =
   26.0275   70.7501  192.3187  522.7764
```

Solution sans les boucles for (fichier fic_for2.m)

fichier fic_for2.m

```
function x = fic_for2(n,m)
x=sum((((exp(1:n))'*ones(1,m)).*(ones(n,1)*log((1:m).^2)))');
```

```
>>  fic_for2(4,5)
ans =
   26.0275   70.7501  192.3187  522.7764
```

Dans ce fichier, nous mettons en œuvre les fonctionnalités du calcul matriciel et celui élément par élément.

Les temps de calcul nécessaire aux fichiers fic_for1 et `fic_for2` sont obtenus par les fichiers scripts `cmp_for1.m` et `cmp_for2.m`.

fichier cmp_for1.m

```
% fonction avec des boucles for
clc
```

```
% initialisation du timer
tic

fic_for1(300,200);
```

```
temps_ecoule1 = toc

>> cmp_for1
temps_ecoule1 =
    0.0658
```

```
>> cmp_for2
temps_ecoule2 =
    1.1105
```

On remarque bien que le calcul vectoriel est nettement mieux qu'avec l'utilisation des boucles `for`.
Le rapport des 2 durées de calcul est alors :

```
>> temps_ecoule2/temps_ecoule1

ans =
   16.8722
```

III.2. Boucle while

La boucle est exécutée tant que la condition qui suit la commande `while` est vraie.

fichier calcul_avec_while.m

```
clc, disp('Avec boucle While')
x = [0.1 0.2 0.5 1.0 1.5 1.9 2.0 3.0 4.0 6.0];
y = [0.95 0.89 0.79 0.70 0.63 0.58 0.56 0.45 0.36 0.28];

% Initialisation des variables
mat_num_a=0;
k=1;

while k<=length(x)
    mat_num_a= mat_num_a+[y(k) x(k);x(k)*y(k) x(k)^2];
    k=k+1;
end

det_mat_num=det(mat_num_a)
```

Nous obtenons le résultat suivant :

```
det_mat_num =
  264.7694
```

III.3. Condition if…else

La syntaxe de la commande `if` est la suivante :

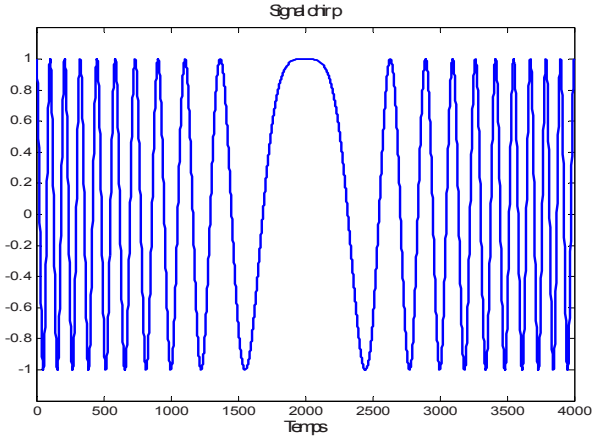
```
if condition1
     Liste1 de commandes
elseif condition2
     Liste2 de commandes
else
     Liste3 de commandes
end
```

Si `Condition1` est vraie, alors `Liste1` est exécutée, sinon et si `condition2` est vérifiée alors il y a exécution de `Liste2`, autrement c'est `Liste3`. Considérons le cas où on peut choisir un signal selon la valeur 1 ou 2 qu'on donne à la variable `signal`, une sinusoïde pour `signal=1`, un sinus cardinal pour `signal=2`, autrement ce sera le signal `chirp` qui sera pris en considération.

fichier choix_signal_if.m

```
clc
signal=3;
t=-2:0.01:1;
if eq(signal,1)
     signal=sin(2*pi*0.5*t)
elseif signal==2
signal=sinc(2*pi*0.5*t);
else
signal=chirp(t,0,1,5);
end
plot(signal)
```

Dans le cas où `signal` vaut 2 ou toute autre valeur différente de 1 ou 2, ce sera le signal `chirp` qui sera tracé. La commande de test d'égalité `eq(signal,1)` consiste à tester l'égalité de la valeur de la variable signal à 1. On peut aisément la remplacer par `isequal(signal, 1)` ou `signal==1`. Avec la valeur `signal=3`, nous avons le tracé du signal `chirp` suivant :



III.4. Condition switch…case

Cette commande permet de basculer d'une liste de commandes vers une autre selon la valeur de l'expression qui se trouve juste après la commande `switch`.

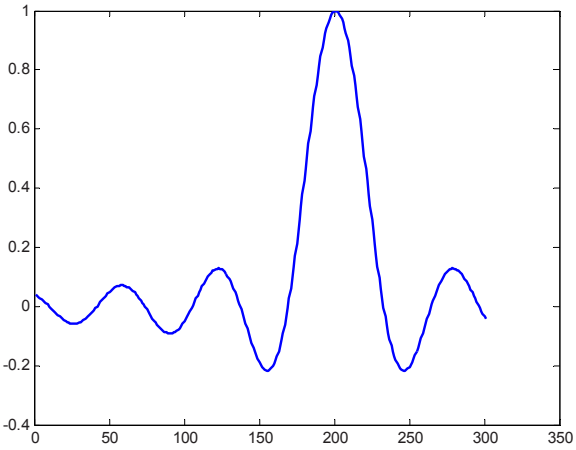
```
switch  expression
     case val_expression1,
           Liste1 de commandes
     case val_expression2
           Liste2 de commandes          ...
     otherwise
           Liste3 de commandes
end
```

Cette boucle ressemble beaucoup à la condition `if`, car dans les 2 cas, on teste les valeurs de l'expression.

Considérons le cas utilisé pour la condition `if`.

choix_signal_switch.m

```
clc, signal=2;
t=-2:0.01:1;
switch signal
    case 1
        signal=sin(2*pi*0.5*t)
    case 2
        signal=sinc(2*pi*0.5*t);
    otherwise
    signal=chirp(t,0,1,5);
end
plot(signal)
```



III.5. Instructions de rupture de séquence

Pour quitter une boucle, nous avons les instructions `break`, `return` et `error('message')`.

- **`break`**

Termine l'exécution d'une boucle. Si plusieurs boucles sont imbriquées, `break` permet de sortir de la boucle la plus proche.

- **`return`**

Cette instruction permet de revenir au fichier M ayant appelé le programme courant ou à la ligne de commande MATLAB.

- **`error('message')`**

Affiche le message spécifié et interrompt l'exécution du programme.

IV. Scripts et fonctions

IV.1. Fichiers fonctions

IV.1.1. Définitions et exemple

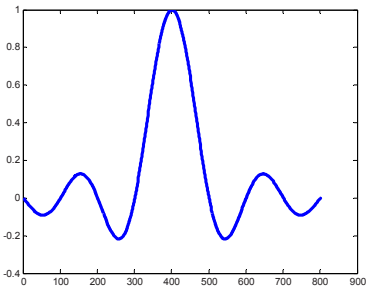
Une fonction est un fichier qu'on sauvegarde dans la liste des chemins de MATLAB avec passage de paramètres dits d'entrée ou d'appel. Elle retourne d'autres paramètres dits de sortie ou de retour.

fonction sinc2.m

```
function y=sinc2(x)
% fonction sinus cardinal
% x : vecteur des abscisses
% y : vecteur des ordonnées
y=(x==0)+sin(x)./((x==0)+x);
```

On définit un angle pour le tracé de la fonction `sinc`, et son appel se fait à l'intérieur de la fonction `plot`.

```
>> x=-4*pi:pi/100:4*pi; plot(sinc2(x), 'LineWidth',3);
```



Les variables à l'intérieur d'une fonction sont locales à celle-ci.

On efface toutes les variables de l'espace de travail et on fait appel à la fonction `sinc2`, comme précédemment.

```
>> clear all
>> alpha=-4*pi:pi/100:4*pi;
>> plot(sinc2(alpha), 'LineWidth',3);
```

Si on invoque la variable `x` :

```
>> x
??? Undefined function or variable 'x'.
```

La variable `x` n'est pas connue dans l'espace de travail. Pour qu'elle le soit, il faut la rendre globale par la commande `global x` à l'intérieur de la fonction.

On peut définir plusieurs variables en tant que globales simultanément par :

```
global x y z
```

On ne peut rendre globale une variable qu'avant son utilisation dans une expression de calcul.

Considérons la fonction suivante dans laquelle on effectue le calcul de l'expression suivante :

```
y = sinc(x)+0.1*randn(size(x))
```

en utilisant 2 variables intermédiaires qu'on déclarera comme globales.

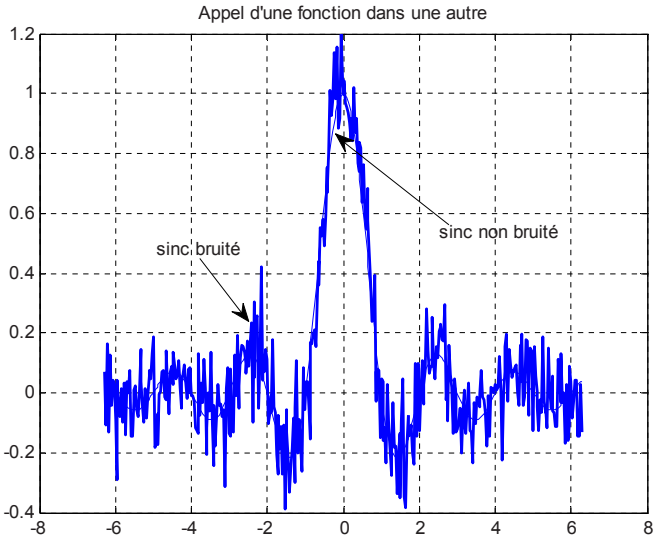
fonction sinc_bruitee

```
function y=sinc_bruite(x)
global y1 y2
y1 = sinc(x);
y2 = 0.1*randn(size(y1))
y=y1+y2;
```

On peut faire appel à une fonction à l'intérieur d'une autre fonction. Ici, on appelle la fonction `sinc` dans la fonction `sinc_bruite`.

fichier var_glob.m

```
clear all, clc
x=-2*pi:pi/100:2*pi;
y=sinc_bruite(x);
plot(y,'LineWidth',2)
title('Appel d''une fonction dans une autre')
grid
```



Comme les variables `y1` et `y2` sont déclarées comme globales dans l'espace de travail et dans la fonction `sinc_bruite`, nous pouvons alors avoir leurs valeurs.

Ici, on affiche les 5 premières valeurs de `y1` et les 3 premières de `y2`.

```
% Appel des variables globales
global y1 y2
y1=y1(1:5)
y2=y2(1:3)

y1 =
   -0.0000   -0.0050   -0.0101   -0.0152   -0.0204

y2 =
   -0.0560   -0.1226    0.0793
```

IV.1.2. Fonctions polymorphes

Une telle fonction retourne différents types de résultats selon le nombre et/ou le type des arguments d'appel ou de retour.

Un exemple est donné par la fonction `abs` qui retourne le module d'un nombre complexe ou le code ASCII des caractères d'une chaîne de caractères.

```
>> x=3+2i;
>> module_x = abs(x)
```

```
module_x =
    3.6056
```

```
>> abs('Casa')
ans =
    67    97   115    97
```

Selon le nombre d'arguments de retour demandés à la fonction, le résultat sera différent.

C'est le cas de la fonction `butter` (filtre de `Butterworth`) qui réalise un type différent du filtre (passe bas, …) et qui retourne les matrices d'état, les pôles et zéros du filtre, etc.

`[B,A]`	=	retourne la fonction de transfert
`butter(N,Wn,'high')`		
`[Z,P,K] = butter(...)`		retourne les pôles et zéros
`[A,B,C,D] = butter(...)`		retourne les matrices d'état A,B,C,D

On se propose d'écrire une fonction qui fait différents types de calculs selon le nombre et le type des arguments d'appel :

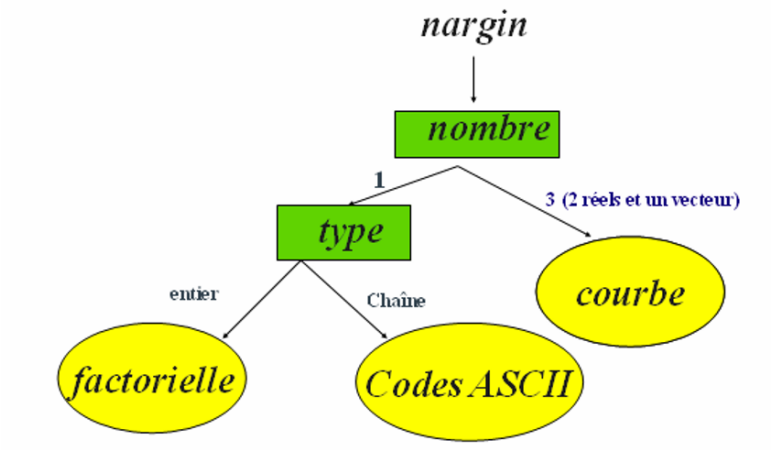
- Factorielle d'un nombre entier ou codes ASCII d'une chaîne,
- Calcul et tracé de la fonction `y=a*x^b` si elle reçoit 3 paramètres dont `a,b`, 2 réels et le troisième, `x` un vecteur d'abscisses.

On teste le nombre d'arguments d'appel ou d'entrée, `nargin`. Si `nargin` est égal à 1, on retourne sa factorielle s'il est entier ou les codes `ASCII` des caractères si c'est une chaîne de caractères.

Si `nargin` vaut 3 (2 réels `a`, `b` et un vecteur `x`), on trace la courbe de $y = a x^2 + b$.

Dans tous les autres cas, on quitte la fonction sans retourner de résultat.

Le nombre d'arguments de retour (de sortie) est contenu dans la variable `nargout`.



Ces tests sont réalisés dans la fonction suivante.

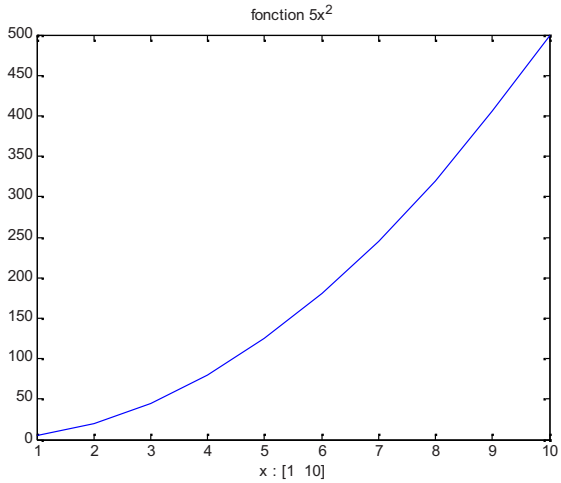
fonction fonct_multiple.m

```
function fonct = fonct_multiple(a,b,x)
% retourne la factorielle d'un nombre entier naturel a
% si elle reçoit un seul argument entier naturel
% calcul la somme des factorielles des 2 paramètres a et b si
elle
% reçoit 2 paramètres d'appel.
% calcul et trace la fonction y=a*x^b si elle reçoit 3
paramètres
% a, b et x.
%
% la factorielle est calculée par la fonction fact.m
utilisant la
% fonction prod.

switch nargin % test du nombre de paramètres d'appel
    case 1
        if ~isstr(a)
        fonct=fact(a);
    else
        fonct=abs(a);
    end
    case 2
        fonct=fact(a)+fact(b);

    case 3
        l=length(x);
        if (l==1)|(isempty(l))
            error ('x doit être un vecteur des abscisses ')
        end
                plot(x,a*x.^b)
title(['fonction ' num2str(a) 'x^{' num2str(b) '}'])
xlabel(['x : [' num2str(x(1)) '  ' num2str(x(length(x))) ']'])
  otherwise
            disp('Unknown method.')
            return
end
```

Les différentes réalisations de la fonction `fonct_multiple` sont résumées ci-après :

```
>> fonct_multiple(5)
ans =
   120
>> fonct_multiple('MATLAB')
ans =
    77    65    84    76    65    66
```

```
>> fonct_multiple(5,2,1:10)
```



On se propose de créer une fonction qui calcule les polynômes de *Chebyshev* de première espèce par les formules suivantes :

Tn(x) = 2xTn − 1(x) − Tn − 2(x), n = 2, 3, …, T0(x) = 1, T1(x) = x.

La fonction suivante réalise ces calculs de façon itérative.

```
function T = ChebT(n)
% Coefficients T of the nth Chebyshev polynomial of the first
kind.
% They are stored in the descending order of powers.
t0 = 1;
t1 = [1 0];
if n == 0
   T = t0;
```

```
elseif n == 1;
   T = t1;
else
   for k=2:n
        T = [2*t1 0] - [0 0 t0];
        t0 = t1;
        t1 = T;
end
end
>> T=chebt(5)
T =
    16     0   -20     0     5     0
```

Sous sa forme récursive, elle a le code suivant :

```
function T=chebt_recc(n)
% Coefficients T of the nth Chebyshev polynomial of the first
kind.
% They are stored in the descending order of powers.
% Edward Neuman
% Department of Mathematics
% Southern Illinois University at Carbondale
(edneuman@siu.edu)
t0 = 1;
t1 = [1 0];
if n == 0
   T = t0;
elseif n == 1;
   T = t1;
else
   T = [2*chebt_recc(n-1) 0]-[0 0 chebt_recc(n-2)];
end
end
```

```
>> T=chebt_recc(5)
T =
    16     0   -20     0     5     0
```

Ces polynômes sont retournés dans l'ordre décroissant de ses coefficients, ordre qu'on peut rétablir par la commande `fliplr`.

```
>> T=fliplr(T)
T =
     0     5     0   -20     0    16
```

La forme itérative est préférable à la forme récursive des fonctions.

```
Clc, disp('Forme itérative:')
tic
T=chebt(5); toc
disp('---')
disp('Forme récursive:'), tic
T=chebt_recc(5);
toc
```

Nous obtenons les temps de calculs suivants, pour ces 2 types de fonctions.

```
Forme itérative :
Elapsed time is 0.000573 seconds.
```

```
---
Forme récursive :
Elapsed time is 0.001531 seconds
```

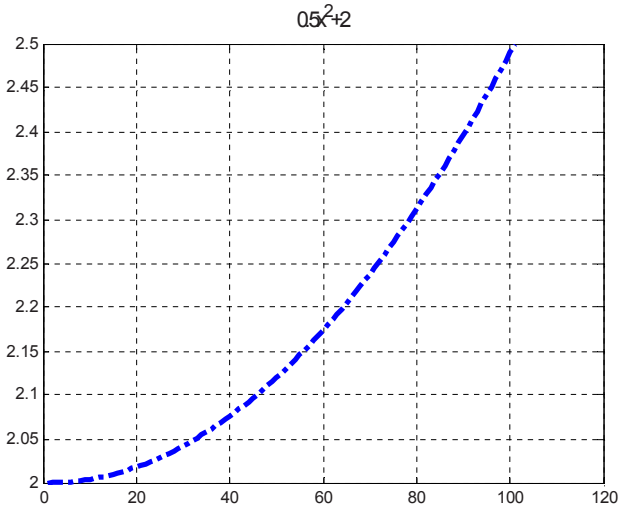
La variable `varargin` est utilisée au sein d'une fonction pour contenir un nombre optionnel et variable d'arguments d'appel.
La variable `nargin` contient le nombre total d'arguments d'appel. Cette fonction effectue la somme ou la différence des 2 premiers arguments d'appel si on lui envoie 2 arguments standards `(a, b)` et un troisième (1 argument optionnel, soit `varargin` {1}) sous forme de la chaîne de caractères 'somme' ou 'diff'.
Si elle reçoit 5 arguments au total (2 standards et 3 optionnels), elle trace la courbe ax^2+b, `x` étant le premier argument d'appel optionnel avec des caractéristiques graphiques spécifiées dans les 2 derniers arguments optionnels.
De même, une fonction peut retourner un nombre variable d'arguments en utilisant la variable `varargout`.

```
function test_vararg(a,b,varargin)

clc
nbre_argin = nargin;
taille_varargin=size(varargin,2);
stdargin = nbre_argin - taille_varargin;

fprintf('Nombre d''arguments d''appel = %d\n', nargin)
fprintf('Entrées standards : %d\n', stdargin)

if isequal(stdargin,2)& isstr(varargin{1});
switch varargin{1}
      case 'somme'
          disp(['Somme =' num2str(a+b)])
      case 'diff'
          disp(['diff =' num2str(a-b)])
end
end

if isequal(stdargin,2)& ~isstr(varargin{1});

plot(a.*varargin{1}.^2+b,'LineStyle',varargin{2},'LineWidth',
varargin{3});
    grid
    title([num2str(a) 'x^2+' num2str(b)])
end
 end
```

On applique les différents cas suivants où l'on obtient successivement la somme, puis la différence des arguments `a` et `p` et enfin la courbe avec un style en pointillés, de largeur 3.

```
>> a=1; b=2; test_vararg(a,b,'somme')
Nombre d'arguments d'appel = 3
Entrées standards : 2
Somme =3
```

```
>> a=1; b=2; test_vararg(a,b,'diff')
Nombre d'arguments d'appel = 3
Entrées standards : 2
diff =-1

>> a=0.5; b=2; test_vararg(a,b,0:0.01:1,'-.',3)
Nombre d'arguments d'appel = 5
Entrées standards : 2
```



IV.1.3. Récursivité des fonctions

La factorielle est récursive du fait que `n ! = n. (n-1) !`

fonction factorielle_recursive.m

```
function fact = factorielle_recursive(N)
% retourne la factorielle d'un nombre entier naturel N
% fonction factorielle récursive fact =

factorielle_recursive(N)
% fact     : valeur retournée de la factorielle de N
% N        : entier naturel pour lequel on calcul la
%            factorielle
if nargin>1;
   help factorielle_recursive
   error('La fonction accepte un nombre entier naturel');
end;
```

```
% cas où n n'est pas un entier naturel
if (fix(N)~=N)| (N<0)
    help factorielle_recursive
   error('N doit être un entier naturel')
end

if N==0
   fact = 1;  % condition d'arrêt
else
   fact = factorielle_recursive(N-1)*N; % relation de
récurrence
end
```

```
>> factorielle_recursive(77)
ans =
  1.4518e+113
```

```
>> factorielle_recursive(6.5)
  retourne la factorielle d'un nombre entier naturel N
  fonction factorielle récursive fact =
factorielle_recursive(N)
  fact     : valeur retournée de la factorielle de N
  N        : entier naturel pour lequel on calcul la
             factorielle

??? Error using ==> factorielle_recursive at 16
N doit être un entier naturel
```

La factorielle est calculée par la fonction `fact.m`.

fichier fact.m

```
function factorielle=fact(N)
% Calcul de la factorielle d'un nombre entier
% naturel en utilisant la fonction prod
% la fonction admet un nombre entier naturel
if nargin>1;
   clc
   help fact
   error('nombre d''arguments incorrect');
end;

if isstr(N)
   clc
   help fact
   error('l''argument doit être entier naturel');
end;
% cas où n n'est pas un entier naturel
if (fix(N)~=N)|(N<0)
```

```
   help fact
   error('N doit être un entier naturel')
end

if N==0
   factorielle = 1;
else
   factorielle=prod(1:N);
end
```

MATLAB dispose de la fonction `factorial` pour calculer la factorielle d'un nombre entier.

Elle peut aussi être calculée avec l'utilisation de la fonction `Gamma`, soit `n*gamma(n)` sauf pour `n=0` pour laquelle on obtient une indétermination.

```
>> factorial(5)

ans =
   120
```

IV.2. Les sous-fonctions

On se propose d'étudier l'évolution des valeurs de suite u_n suivante :

$$u_n = \frac{3+\frac{\cos n}{n^2}}{4(1+\frac{2}{n}+\frac{1}{n^2})+\frac{\sin 3n}{n^2}}$$

Pour chaque valeur de l'indice n, on réalise les mêmes opérations de calcul des 3 expressions $3+\frac{\cos n}{n^2}$, $4(1+\frac{2}{n}+\frac{1}{n^2})$ et $\frac{\sin 3n}{n^2}$.

Il est alors judicieux de faire appel à la même routine de calcul, appelée sous-fonction (fonction de fonction). Dans ce qui suit, les routines `numer`, `den1` et `den2` sont définies comme des sous-fonctions qui calculent les 3 expressions précédentes.

fichier subfunction.m

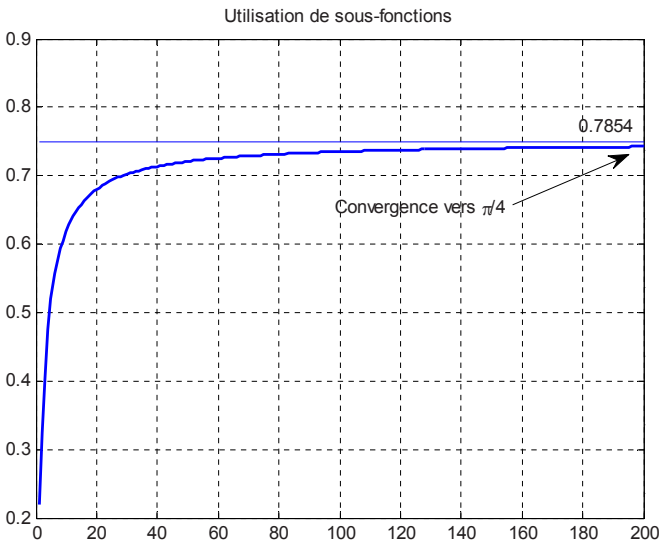
```
function u = subfunction(n)
% calcul de u par appel de 3 sous-fonctions
%num, den1 et den2
u=num(n)./(den1(n)+den2(n));
% sous-fonction num
function numer=num(n)
numer=3+cos(n)./(n.^2);
% sous-fonction den1
function d1= den1(n)
```

```
d1=4*(1+2./n+1./(n.^2));
% sous-fonction den2
function d2=den2(n)
d2=sin(3*n)./(n.^2);
```

Dans le fichier suivant, on calcule l'expression précédente, en utilisant la sous-fonction `subfunction` pour `n` allant de 1 à 200.

fichier use_subfunction.m

```
clc
clear all
n=1:200;
y=subfunction(n);

plot(n,y)
% valeur de convergence 3/4
x=3*ones(size(y))/4;

hold on
plot(n,x)
gtext('Convergence vers \pi/4')
```



La valeur de l'expression u_n converge vers $\frac{\pi}{4} \approx 0.7854$.

V. Conseils de programmation sous MATLAB

Il est utile de suivre certains conseils de programmation dont, entre autres, les suivants :
- choisir des noms significatifs pour les variables et les fonctions,
- documenter les fonctions pour l'aide en ligne et la clarté,
- vérifier le nombre et le type des arguments d'appel et de retour des fonctions,
- n'utiliser les boucles `for` et `while` ainsi que le `if` que si nécessaire, elles peuvent être souvent remplacées avantageusement par des opérations et fonctions vectorielles ou matricielles,
- utiliser des indentations dans les instructions structurées (`for`, `while`, `if`) lorsqu'on est obligé de les utiliser.
- Préférer les formes itératives des fonctions aux formes récursives.

Lors de la programmation, il est utile parfois d'ignorer un bloc d'instructions afin de localiser les erreurs.
Pour ignorer temporairement une partie d'un programme, lors de la phase de mise au point par exemple, on pourra utiliser l'une des méthodes proposées ci-dessous :

```
if 0
....
instructions à ignorer
....
end
```

ou

```
while 0
....
instructions à ignorer
....
end
```

- **fichiers P-code**
  Il est parfois utile de cacher le code de certains fichiers. Pour transformer le fichier factorielle_recursive2.m en fichier P, nous utilisons la commande suivante :

```
>> pcode factorielle_recursive2
```

La commande `ls` d'`unix`, ou `dir` de `DOS`, retourne le seul fichier d'extension `.p`.

```
>> ls *.p
factorielle_recursive2.p
```

Si on demande le code par la commande `type`, la réponse est que ce fichier est du type `P-file`.

```
>> type factorielle_recursive2
'factorielle_recursive2' is a P-file.
```

L'exécution de ce fichier donne le même résultat que le fichier M d'origine.

```
>> factorielle_recursive2(5)
ans =
   120
```

Nous pouvons voir la taille du même fichier, avant et après sa transformation en P-code.

```
>> r = dir('factorielle_recursive2.p')
r =
       name: 'factorielle_recursive2.p'
       date: '27-juil.-2009 23:46:59'
       bytes: 246
       isdir: 0
       datenum: 7.3398e+005
```

```
>> r = dir('factorielle_recursive.m')
r =
       name: 'factorielle_recursive.m'
       date: '27-juil.-2009 23:46:37'
       bytes: 680
       isdir: 0
       datenum: 7.3398e+005
```

Nous créons, ci-après, la fonction qui déterminera si un nombre est entier ou pas.

```
function k = isinteg(x);
% Check whether or not x is an integer number.
% If it is, function isint returns 1 otherwise it returns 0.
if abs(x - round(x)) < realmin
k = 1; else k = 0; end
```

On teste si la différence entre ce nombre et sa partie entière est inférieure à la valeur `realmin`[10] qui vaut 2.2251e-308.

```
>> isinteg(6)
ans =
     1
```

Le chiffre 6 auquel on ajoute 10^{-10} n'est plus entier.

```
>> >> isinteg(6.0000000001)
ans =
     0
```

MATLAB R2009 possède la fonction isinteger pour réaliser cette opération. Le même chiffre 6 ne sera pas vu comme entier car par défaut, il appartient à la classe `double`.

```
>> isinteger(6)
ans =
     0
```

Pour qu'il soit vu comme un entier, il suffit de transformer son type en entier par `int8`(6), codé sur 8 bits, par exemple.

```
>> isinteger(int8(6))
ans =
     1
```

Dans ce cas, ce chiffre est bien vu comme non entier.

VI. Déboguage des fichiers

Exemple d'une session de mise au point d'un programme

Nous utiliserons comme exemples de programmes pour une session de mise au point, des fonctions réalisées précédemment. Il s'agit de la fonction
`factorielle_recursive.m`
Nous pouvons utiliser la commande `dbtype` pour afficher les lignes de code numérotées de ces fonctions.

```
>> dbtype chebt
1     function T = ChebT(n)
2     % Coefficients T of the nth Chebyshev polynomial of the
      first kind.
3     % They are stored in the descending order of powers.
4     t0 = 1;
5     t1 = [1 0]
6     if n == 0
7     T = t0;
8     elseif n == 1;
9     T = t1;
10    else
11    for k=2:n
12    T = [2*t1 0] - [0 0 t0];
13    t0 = t1; t1 = T;
15    end
16    end
```

- Nous pouvons fixer des points d'arrêt dans les différentes fonctions, nous utiliserons pour cela la commande `dbstop`.

La commande suivante fixe un point d'arrêt à la ligne 10 de la fonction `chebT.m`

```
>> dbstop at 10 in chebt
```

```
1   function T = ChebT(n)
2   % Coefficients T of the nth Chebyshev polynomial of the first kind.
3   % They are stored in the descending order of powers.
4   t0 = 1;
5   t1 = [1 0];
6   if n == 0
7   T = t0;
8   elseif n == 1;
9   T = t1;
10  else
11  for k=2:n
12  T = [2*t1 0] - [0 0 t0];
13  t0 = t1;
14  t1 = T;
15  end
16  end
```

On remarque le point d'arrêt à la ligne 10.

L'exécution de la fonction `chebt` s'interrompt à la rencontre de ce point d'arrêt.

```
1   function T = Chebt(n)
2   % Coefficients T of the nth Chebyshev polynomial of the first kind.
3   % They are stored in the descending order of powers.
4   t0 = 1;
5   t1 = [1 0];
6   if n == 0
7   T = t0;
8   elseif n == 1;
9   T = t1;
10  else
11  for k=2:n
12  T = [2*t1 0] - [0 0 t0];
13  t0 = t1;
14  t1 = T;
15  end
16  end
```

```
>> chebt(5)
10  else
K>>
```

On se retrouve alors dans le mode "`debug`" signalé par l'invite " K>>". Nous pouvons, à ce niveau, utiliser les commandes de mise au point (`dbstep`, `dbcont`, `dbstack`, `dbstop`, etc.).

La commande `dbstep` permet de progresser ligne par ligne dans l'exécution de la fonction `ppcm`.

```
K>> dbstep
11  for k=2:n
K>>
```

Nous pouvons consulter la liste des variables locales à la fonction, définies avant la ligne 11.

```
K>> whos
  Name        Size              Bytes  Class     Attributes
  n           1x1                   8  double
  t0          1x1                   8  double
  t1          1x2                  16  double
```

Les variables peuvent être consultées et modifiées.

```
K>> t1
t1 =
     1     0
```

Les variables définies dans l'espace de travail ne sont pas visibles pour la fonction `chebt`.

```
K>> x
??? Undefined function or variable 'x'.
```

Seules les variables définies dans la fonction `chebt` ne sont visibles dans on espace de travail.
La variable `x` étant déjà définie dans l'espace de travail MATLAB comme une matrice aléatoire.

Pour remonter de l'espace de travail de la fonction à celui de MATLAB, on utilise la commande `dbup`.

```
K>> dbup
In base workspace.
```

```
K>> whos
  Name        Size              Bytes  Class     Attributes

  ans         1x6                  48  double
  x           4x4                 128  double
```

Pour redescendre à l'espace de travail de la fonction, on utilise `dbdown`.

```
K>> dbdown
In workspace belonging to chebt at 11
```

```
K>> whos
  Name        Size              Bytes  Class     Attributes
  n           1x1                   8  double
  t0          1x1                   8  double
  t1          1x2                  16  double
```

On retrouve uniquement les variables de l'espace de travail de la fonction.

Pour continuer l'exécution de la fonction jusqu'à la fin de celle-ci ou jusqu'au prochain point d'arrêt, on utilisera la commande `dbcont`.

```
K» dbcont
ans =
    16     0   -20     0     5     0
```

On retrouve alors l'espace de travail de base avec les variables définies dans ce dernier, après la commande `dbup`.

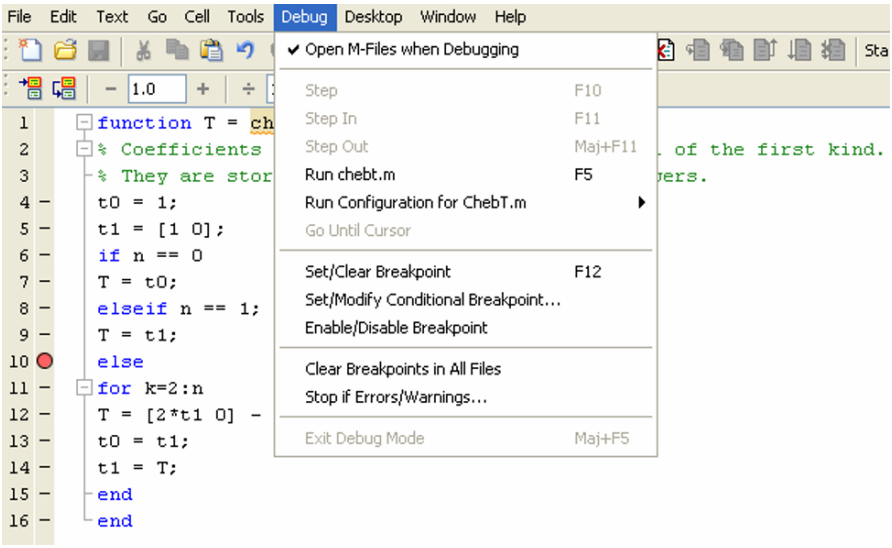
```
» who
  Name        Size              Bytes  Class     Attributes
  ans         1x6                  48  double
  x           4x4                 128  double
```

On retrouve les variables définies auparavant, avant d'entrer dans le mode `debug`, dans l'espace de travail MATLAB.

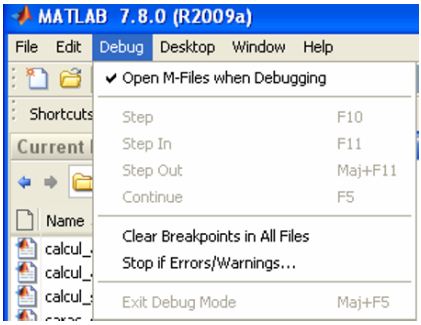
On peut réaliser les commandes précédentes en mode `debug` par les options du menu `debug` de la fenêtre de l'éditeur où sont affichées les instructions de la fonction `chebt` à débuggeur.

Le point d'arrêt est signalé par un cercle rouge à la ligne 10.

Par le menu `debug`, on peut exécuter le fichier, mettre ou supprimer un point d'arrêt, permettre ou pas un point d'arrêt, etc.



La fenêtre principale de MATLAB contient aussi le menu `debug`.

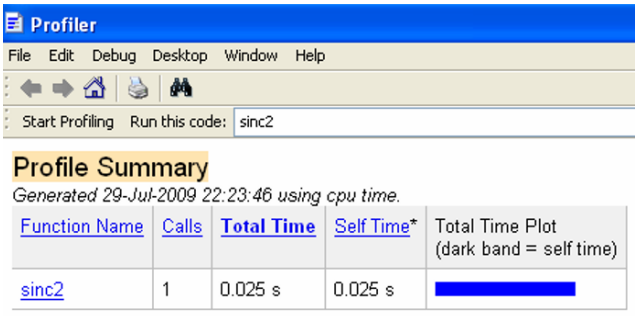


VII. Le profiler

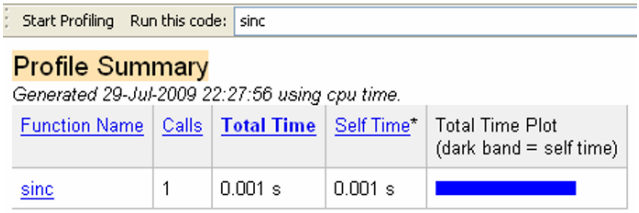
Le principe du profiler consiste à étudier et améliorer les performances d'un programme.
Par la commande `profview`, on ouvre une interface HTML pour lancer le profiler.
Pour étudier les performances d'un programme, MATLAB fournit une interface utilisateur (GUI). Améliorer les performances d'un programme consiste par exemple à déterminer les lignes de programme qui consomment plus de temps d'exécution, afin de rechercher à les améliorer.
Nous allons étudier le profiler sur un programme qui utilise la récursivité plutôt que l'itération.
Pour ouvrir le profiler, on peut, soit utiliser la commande `profile` qui possède beaucoup d'options, ou choisir l'option `Profiler` du menu `Desktop` de la fenêtre principale de MATLAB.
On peut aussi utiliser le menu `Tools … Open profiler` de la fenêtre de l'éditeur une fois qu'on a ouvert ce fichier.

Un autre moyen est d'exécuter la commande `profview` de MATLAB qui affiche l'interface HTML du profiler.

Nous allons étudier le fichier `sinc2.m` pour le comparer à la version disponible dans MATLAB sans aucune expression logique pour lever l'indétermination.



Nous avons le résultat suivant pour `sinc2` :


Start Profiling Run this code: sinc

Profile Summary

Generated 29-Jul-2009 22:27:56 using cpu time.

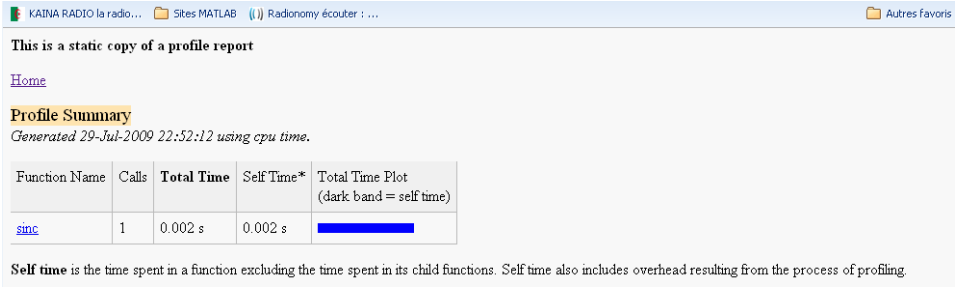
Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
sinc	1	0.001 s	0.001 s	

La fonction `sinc2.m` met 25 fois plus de temps que `sinc` m, disponible dans MATLAB, à cause des tests relationnels de l'instruction :

```
y=(x==0)+sin(x)./((x==0)+x);
```

```
profile on
y=sinc(-2*pi:pi:2*pi);
profile viewer
profsave(profile('info'),'profile_results')
```

Le profiler aboutit au rapport suivant avec la fonction `sinc` de MATLAB.


KAINA RADIO la radio... Sites MATLAB Radionomy écouter : ... Autres favoris

This is a static copy of a profile report

Home

Profile Summary

Generated 29-Jul-2009 22:52:12 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
sinc	1	0.002 s	0.002 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

En utilisant `sinc2.m`, nous trouvons les résultats suivants :


file:///C:/Documents%20and%20Settings/MARTAJ/Mes%20documents/MATLAB/Chapitres%20prochains/Programmation

KAINA RADIO la radio... Sites MATLAB Radionomy écouter : ... Autres

This is a static copy of a profile report

Home

Profile Summary

Generated 29-Jul-2009 22:46:51 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
sinc2	1	0.129 s	0.129 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the process of profiling.

Avec les commandes `profile on` et `profile off`, on démarre et on arrête respectivement le profiler.

La commande `profsave` sauvegarde le rapport du profiler sous le format HTML.

La commande de syntaxe

```
S = profile('INFO')
```

suspend le profiler et retourne les caractéristiques du profiler dans la structure S suivante :

```
S =
      FunctionTable: [1x1 struct]
      FunctionHistory: [2x0 double]
      ClockPrecision: 3.5714e-010
      ClockSpeed: 2.8500e+009
      Name: 'MATLAB'
      Overhead: 0
```

Nous avons, entre autres, la précision et la vitesse d'horloge du CPU.
Le tableau de structures `FunctionTable` contient des statistiques pour chaque fonction appelée.

```
>> SF=S.FunctionTable

SF =
          CompleteName: [1x64 char]
```

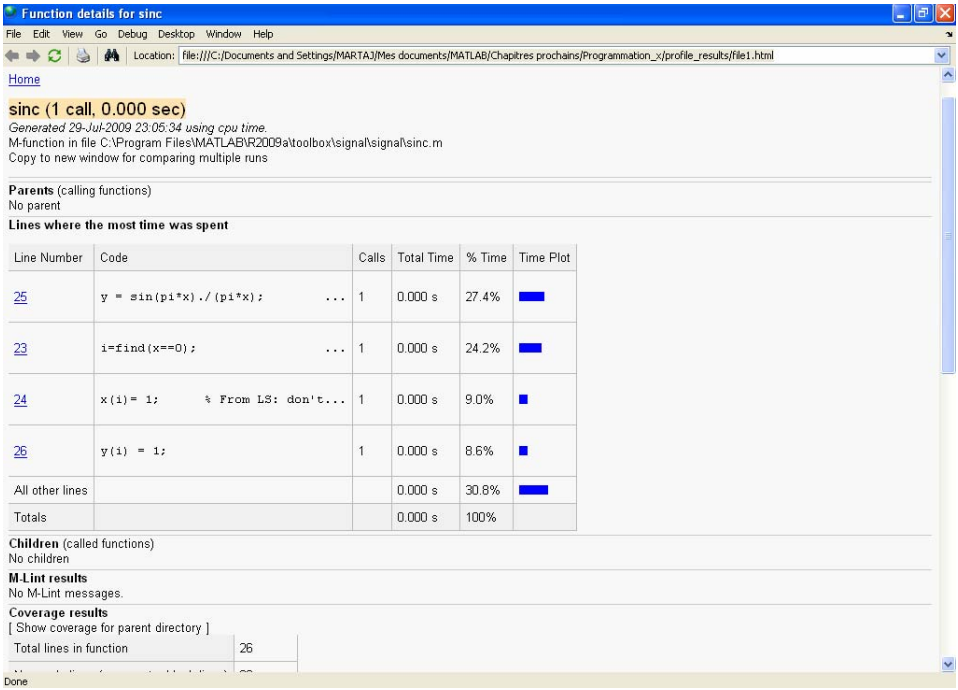
```
          FunctionName: 'sinc'
          FileName: [1x59 char]
          Type: 'M-function'
          Children: [0x1 struct]
          Parents: [0x1 struct]
          ExecutedLines: [4x3 double]
          IsRecursive: 0
          TotalRecursiveTime: 0
          PartialData: 0
          NumCalls: 1
          TotalTime: 1.3642e-004
```

La seule fonction appelée est la fonction `sinc` qui ne contient pas de récursivité (`Isrecursive=0`), de type fichier M (`Type : M-function`), de temps de récursivité nul (`TotalRecursiveTime: 0`), avec un seul appel de la fonction `sinc`.

Le champ `CompleteName` contient le nom de la fonction et le répertoire où elle se trouve, celui de la boite à outils « `Signal Processing Toolbox` ».

```
>> SF.CompleteName
ans =
C:\Program
Files\MATLAB\R2009a\toolbox\signal\signal\sinc.m>sinc
```

Dans le répertoire `profile_results`, nous trouvons le fichier HTML suivant, qui détaille la durée d'exécution de chaque fonction et commande à l'intérieur de la fonction `sinc`.



Function details for sinc

File Edit View Go Debug Desktop Window Help

Location: file:///C:/Documents and Settings/MARTAJ/Mes documents/MATLAB/Chapitres prochains/Programmation_x/profile_results/file1.html

Home

sinc (1 call, 0.000 sec)

Generated 29-Jul-2009 23:05:34 using cpu time.
M-function in file C:\Program Files\MATLAB\R2009a\toolbox\signal\signal\sinc.m
Copy to new window for comparing multiple runs

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
25	y = sin(pi*x)./(pi*x); ...	1	0.000 s	27.4%	
23	i=find(x==0); ...	1	0.000 s	24.2%	
24	x(i)= 1; % From LS: don't...	1	0.000 s	9.0%	
26	y(i) = 1;	1	0.000 s	8.6%	
All other lines			0.000 s	30.8%	
Totals			0.000 s	100%	

Children (called functions)
No children

M-Lint results
No M-Lint messages.

Coverage results
[Show coverage for parent directory]

Total lines in function	26

Done

VIII. Les fichiers de données

Les données peuvent être stockées dans différents types de fichiers :

- fichiers M pour la sauvegarde d'instructions avant leur utilisation dans des programmes
- fichiers MAT ou des fichiers textes et binaires définis par l'utilisateur.

Données dans un fichier M (data.m)

```
% définition de matrices et expression de la solution de A x
= B

A = [1 2 3
     4 5 6];
B = [3 4]';
x= inv(A'*A)*A'*B;
```

Données dans un fichier MAT

Une fois qu'une variable est définie, on pourra utiliser la commande `save` pour la sauvegarder dans un fichier MAT. Sa restauration sera réalisée par la commande `load`.

```
clc, clear all, alpha = -2*pi:pi/100 :2*pi;
x = rand(4); alpha ;
save fic_mat x alpha
```

Sauvegarde de la variable `x` et `alpha` dans le fichier MAT `fic_mat.mat`.

```
>> load fic_mat
>> who
Your variables are:
alpha  ans    x
```

```
>> whos
  Name        Size              Bytes  Class     Attributes

  alpha       1x401              3208  double
  ans         1x401              3208  double
  x           4x4                 128  double
```

Dès qu'on ouvre le fichier `mat`, les variables qu'il contient sans dans l'espace de travail. La variable `x` est bien une matrice 4x4 et `alpha` un vecteur de 401 éléments.
On peut sauvegarder les variables dans une structure.

```
>> S1.x=[1 2 ;4 7] ;
>> S1.alpha=-4*pi:pi/100:4*p
```

Les variables *x* et `alpha` sont cette fois sauvegardées dans la structure `S1`.

```
S1 =
  x: [2x2 double]
  alpha: [1x801 double]
>> S1.x
ans =
     1     2
     4     7

>> S1.alpha(1:5)
ans =
  -12.5664  -12.5350  -12.5035  -12.4721  -12.4407
```

Dans SIMULINK, le bloc « `to Workspace` » propose en premier lieu la sauvegarde dans une structure puis dans un tableau (`Array`).
La commande `save` permet aussi la sauvegarde dans un fichier ASCII avec la syntaxe suivante :

```
>> alpha = [-pi -pi/4 0 pi/4 pi];
>> save fict3.txt x alpha -ascii -double -tabs
```

```
1.0000000000000000e+000     2.0000000000000000e+000
4.0000000000000000e+000     7.0000000000000000e+000

-3.1415926535897931e+000    -7.8539816339744828e-001
0.0000000000000000e+000     7.8539816339744828e-001
3.1415926535897931e+000
```

x

alpha

Ce fichier `text` peut être aussi ouvert par la commande :

```
>> open fict3.txt
```

Ces fichiers peuvent être écrits et lus par des commandes type langage C, telles `fprintf` et `fscanf`, etc.

IX. Les commandes et outils de développement

IX.1. Commandes de gestion d'environnement

- **`quit`**

Permet de quitter la session de travailler et de fermer MATLAB.

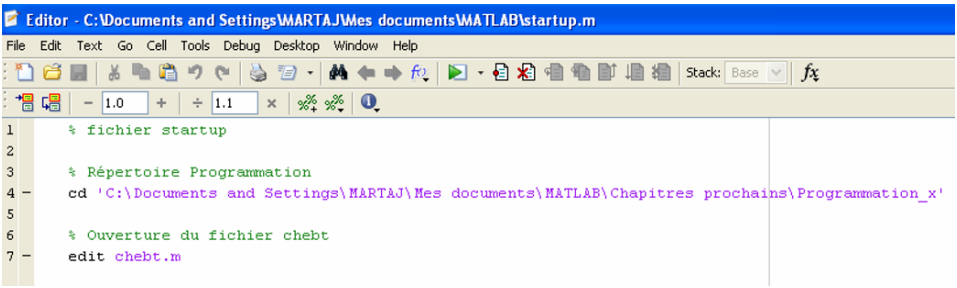
- **`Exit`**

Même fonction que `quit`.

- **`Fichier startup.m`**

Le fichier `startup.m` est exécuté automatiquement à chaque ouverture de MATLAB. Il est très utile dans l'industrie lorsqu'on a besoin d'ouvrir beaucoup d'autres fichiers d'initialisation, comme les paramètres qui interviennent dans une série de mesures (paramètres de régulation, des capteurs, etc.)

Nous avons créé un fichier dans lequel on se met automatiquement dans un répertoire de travail spécifié et qui ouvre un script (fonction `chebt`). C'est utile lorsqu'on n'a pas fini le code d'un programme.



- **clc**

Efface l'écran de MATLAB mais les variables sont toujours présentes dans l'espace de travail. Cette commande est utile au début de tout programme pour plus de lisibilité.
La commande `home` réalise la même fonction en mettant le prompt en haut et à gauche de l'écran.

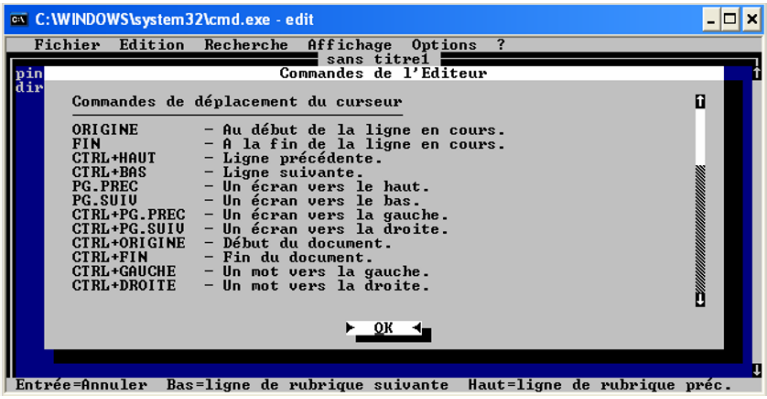
- **Dos**

Exécute les commandes DOS et retourne les résultats.

```
>> [s, w] = dos('dir')
s =
     0
w =
 Le volume dans le lecteur C n'a pas de nom.
 Le num,ro de s,rie du volume est 7041-B406
 R,pertoire de C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Analyse num,rique_x

26/07/2009  05:38    <REP>          .
26/07/2009  05:38    <REP>          ..
24/07/2009  06:38            20ÿ302 fsolve_syst_nl.mdl
24/07/2009  18:13            20ÿ198 fsolve_syst_nl2.mdl
24/07/2009  18:32               261 sol_syst_graph.m
               3 fichier(s)           40ÿ761 octets
               2 R,p(s)  21ÿ377ÿ597ÿ440 octets libres
```

On peut utiliser directement la commande `dir` ou `ls` qui retourne peu de paramètres de retour.

```
>> dir
.                    fsolve_syst_nl.mdl   sol_syst_graph.m
..                   fsolve_syst_nl2.mdl
>> dos('notepad file.m &')
```



Ouvre une fenêtre de l'éditeur du DOS.

- **system**

Exécute les commandes système et retourne les résultats.

```
>> [status,result] = system('dir')
status =
     0
result =
 Le volume dans le lecteur C n'a pas de nom.
 Le num,ro de s,rie du volume est 7041-B406

 R,pertoire de C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Programmation_x

29/07/2009  02:09    <REP>          .
29/07/2009  02:09    <REP>          ..
25/07/2009  05:09               292 calcul_avec_for.m
25/07/2009  17:16               315 calcul_avec_while.m
25/07/2009  05:12               189 calcul_sans_for.m
27/07/2009  00:04                98 carac_spec.m
28/07/2009  22:20               283 chebt.m
```

- **unix**

Permet l'exécution des commandes Unix.

```
>> [s,w] = unix('MATLAB')
```

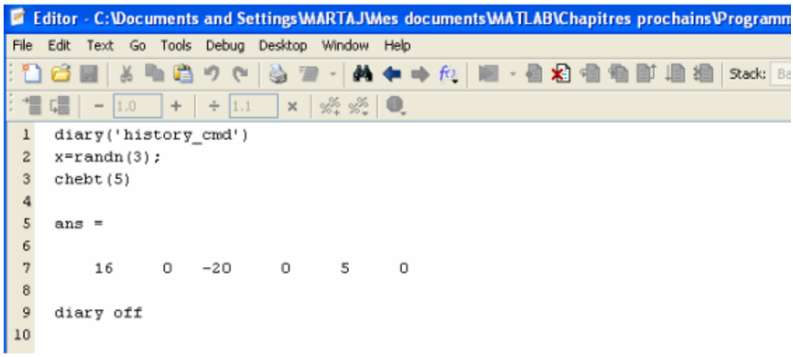
Lance l'exécution de MATLAB.

```
>> pwd
ans =
C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Programmation_x
```

- **diary**

```
>> diary on
>> diary('history_cmd')
>> x=randn(3);

>> chebt(5)

ans =

    16     0   -20     0     5     0

>> diary off
```

Il y a création du fichier `history_cmd` dans lequel sont sauvegardées toutes les commandes exécutées à partir de `diary` on jusqu'à `diary off`.



- **`format`**

Permet de spécifier le format d'affichage des nombres. Ceci n'affecte pas la précision du résultat des calculs. Par défaut, MATLAB affiche les nombres en format court, `short`.

```
>> format long

>> pi
ans =
   3.141592653589793
```

```
>> format short e
>> pi
ans =
  3.1416e+000
```

```
>> format short eng % notation ingénieur
>> exp(100.78766)
ans =
    59.0914e+042
```

```
>> format hex % format hexadécimal
>> pi
ans =
   400921fb54442d18
```

```
>> format rat % format rationnel
```

```
>> pi
ans =
     355/113
```

IX.2. Commandes d'aide à l'utilisation de MATLAB

La commande `help` seule, affiche des quelques types d'aide, par le nom d'un répertoire où l'on peut obtenir cette aide.

- **help**

`help «nom_fonction»` donne de l'aide sur cette fonction.

Cette aide est constituée des lignes de commentaires, sans espace, qui suit la définition de cette fonction.

```
>> help
HELP topics:

Mes documents\MATLAB            - (No table of contents file)
matlab\general                  - General purpose commands.
matlab\ops                      - Operators and special characters.
matlab\lang                     - Programming language constructs.
matlab\elmat                    - Elementary matrices and matrix manipulation.
matlab\randfun                  - Random matrices and random streams.
matlab\elfun                    - Elementary math functions.
matlab\specfun                  - Specialized math functions.
matlab\matfun                   - Matrix functions - numerical linear algebra.
matlab\datafun                  - Data analysis and Fourier transforms.
matlab\polyfun                  - Interpolation and polynomials.
matlab\funfun                   - Function functions and ODE solvers.
```

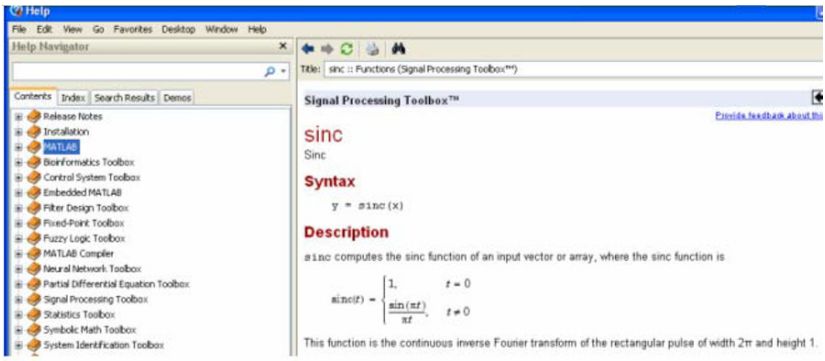
```
>> help sinc2
% aide sur la fonction sinc2.m fonction sinus cardinal
  x : vecteur des abscisses
  y : vecteur des ordonnées
```

La commande `helpwin` donne les mêmes résultats, ainsi que `docsearch`.

- **doc**

`doc « fonction »` ouvre la fenêtre contenant l'aide de MATLAB concernant cette fonction.

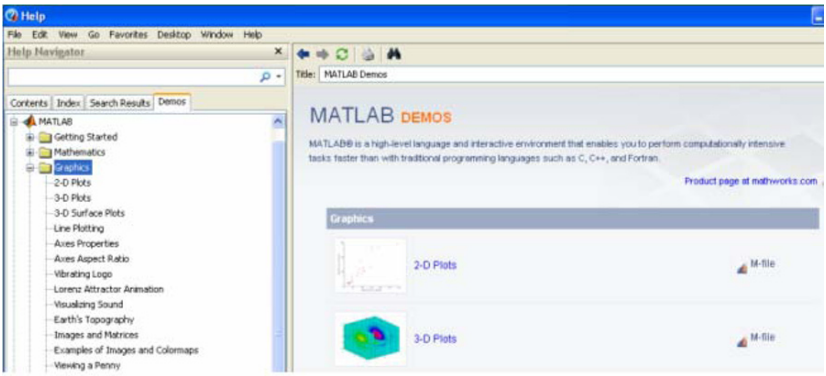
```
>> doc sinc
```



- **demo**

Donne les démos disponibles sur un thème donné de MATLAB.

```
>> demo matlab graphics
```




```
>> demo toolbox signal % démos de « Signal Processing
Toolbox ».
```

- **lookfor**

Recherche toutes les expressions MATLAB contenant un certain mot-clé.
Pour rechercher tout ce qui contient le mot-clé « `chev` », on utilise la commande suivante :

```
>> lookfor cheb
```

```
chebt                          - Coefficients T of the nth Chebyshev
polynomial of
                                 the first kind.
chebt_recc                     - Coefficients T of the nth Chebyshev
polynomial
                                 the first kind.
chebyPoly_atan_fixpt           - Calculate arctangent using Chebyshev
polynomial
                                 approximation
chebyPoly_atan_fltpt           - Calculate arctangent using Chebyshev
polynomial
                                 approximation
poly_atan2                     - Calculate the four quadrant inverse
tangent via Chebyshev polynomial
cheb1ap                        - Chebyshev Type I analog lowpass filter
prototype.
cheb1ord                       - Chebyshev Type I filter order selection.
cheb2ap                        - Chebyshev Type II analog lowpass filter
prototype.
cheb2ord                       - Chebyshev Type II filter order
selection.
chebwin                        - Chebyshev window.
cheby1                         - Chebyshev Type I digital and analog
filter design.
cheby2                         - Chebyshev Type II digital and analog
filter design.
fdcheby1                       - Chebyshev Type I Module for filtdes.
fdcheby2                       - Chebyshev Type II Module for filtdes.
```

Nous retrouvons dans la liste, les 2 fonctions que nous avons créées, `chebt.m` et `chebt_recc.m`, la plupart étant des fonctions de développement de filtres de `Tchebychev`.

- **web**

```
>>     web('http://www.mathworks.com', '-new');
```

Ouvre le site Web de `Mathworks` dans une nouvelle fenêtre.

IX.3. Gestion des répertoires

- **which**

`which « fonct »` donne le répertoire qui contient cette fonction.

```
>> which sinc2
C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Programmation_x\sinc2.m
```

- **cd**

La commande `cd` donne le nom du répertoire de travail courant.

```
>> cd
C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Programmation_x
```

On peut monter ou descendre d'un niveau par rapport au répertoire courant.

```
>> cd ..
>> cd
C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains
```

On peut se déplacer de plusieurs niveaux en même temps, comme ici de 2 niveaux :

```
>> cd ../..
>> cd
C:\Documents and Settings\MARTAJ\Mes documents
```

On peut aussi spécifier directement le nom du répertoire auquel on veut se déplacer.

```
>> cd ('C:\Documents and Settings\MARTAJ\')
>> cd
C:\Documents and Settings\MARTAJ
```

La commande `pwd` donne le même résultat.

- **dir, ls**

Affiche le contenu du répertoire courant.

```
>> ls
.                       fic_for2.m
..                      fic_mat.mat
calcul_avec_for.m       fict3
calcul_avec_while.m     fict3.txt
calcul_sans_for.m       fonct_multiple.m
```

```
chebt.m                    isinteger2.m
chebt_recc.asv             newstruct.mat
Desktop Tools and Development Environment
```

La commande `dir` donne le même résultat.

- **What**

```
W = what('nom_repertoire')
```

Cette commande retourne le contenu du répertoire sous forme d'une structure.

```
>> W=what('C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Analyse numérique_x')

W =

        path: [1x93 char]
           m: {'sol_syst_graph.m'}
         mat: {0x1 cell}
         mex: {0x1 cell}
         mdl: {2x1 cell}
           p: {0x1 cell}
     classes: {0x1 cell}
    packages: {0x1 cell}
```

Pour avoir la liste des fichiers M ou des modèles SIMULINK :

```
>> W.m
ans =
    'sol_syst_graph.m'
>> W.mdl
ans =

    'fsolve_syst_nl.mdl'
    'fsolve_syst_nl2.mdl'
```

- **path**

Liste tous les répertoires du chemin de recherche de MATLAB.

```
>> path
  MATLABPATH

 C:\Documents and Settings\MARTAJ\Mes documents\MATLAB
 C:\Program Files\MATLAB\R2009a\toolbox\matlab\general
…
C:\Program Files\MATLAB\R2009a\toolbox\shared\optimlib
C:\Program Files\MATLAB\R2009a\toolbox\symbolic
```

- **addpath**

Ajoute un répertoire dans la liste des chemins de recherche de MATLAB.

```
>> addpath ('C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains')

>> path
   MATLABPATH
 C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains
 C:\Documents and Settings\MARTAJ\Mes documents\MATLAB
```

Le répertoire est ajouté à la liste des chemins de recherche de MATLAB.

- **savepath**

Sauvegarde les chemins de recherche de MATLAB.

```
>> savepath 'C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\pathdef.m')
```

```
>> type pathdef

function p = pathdef
%PATHDEF Search path defaults.
%   PATHDEF returns a string that can be used as input to
MATLABPATH
%   in order to set the path.

%   Copyright 1984-2007 The MathWorks, Inc.
%   $Revision: 1.4.2.2 $ $Date: 2007/06/07 14:45:14 $
% DO NOT MODIFY THIS FILE.  IT IS AN AUTOGENERATED FILE.
% EDITING MAY CAUSE THE FILE TO BECOME UNREADABLE TO
% THE PATHTOOL AND THE INSTALLER.

p = [...
%%% BEGIN ENTRIES %%%
     'C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains;', ...
     matlabroot,'\toolbox\matlab\general;', ...
     matlabroot,'\toolbox\matlab\ops;', ...
```

- **matlabroot**

Affiche le répertoire racine.

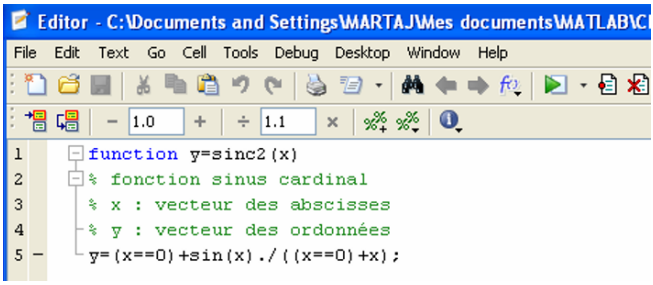
```
>> matlabroot
ans =
C:\Program Files\MATLAB\R2009a
```

X. Editeur de fichiers M

Nous nous proposons d'étudier quelques menus de la fenêtre de l'éditeur de fichiers M. Pour ouvrir cet éditeur, on utilise la commande `edit « nom_fichier ».`

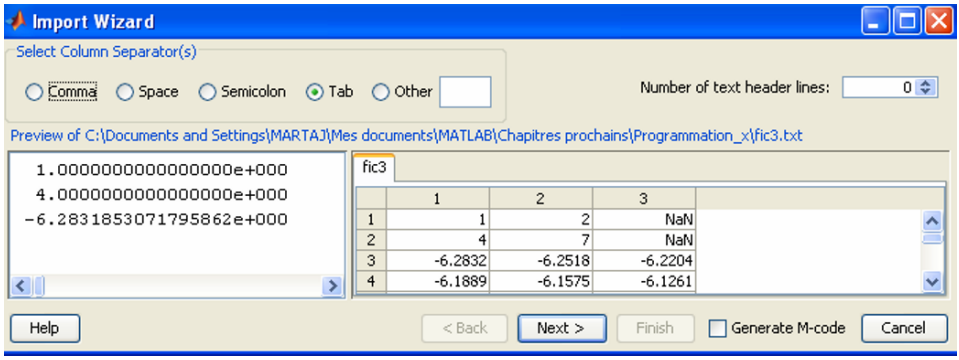
La commande `edit` seule ouvre l'éditeur avec un fichier vide, nommé `Untitled.m`

```
>> edit chebt
```



- **`file ... Import Data ...`**

Permet l'importation de données, d'un fichier `txt` ou `mat` comme `fic_mat.mat`



On peut choisir le type de séparateur (virgule, tabulation, …).
On peut générer le code d'une fonction (`importfile.m`) de lecture de ce fichier, que l'on peut intégrer et appeler d'un autre fichier.

- **`file ... Save Workspace AS ...`**

Les variables de l'espace de travail sont sauvegardées par défaut dans le fichier `matlab.mat` mais on peut choisir un autre nom de fichier.

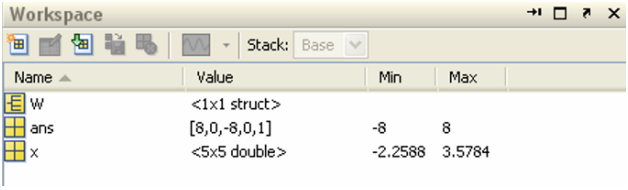
Nous exécutons les commandes suivantes et nous observons les variables dans la fenêtre `Workspace` (espace de travail).

```
>> chebt(4)
ans =
     8     0    -8     0     1
```

```
>> x=randn(5);
>>    W    =    what('C:\Documents    and    Settings\MARTAJ\Mes
documents\MATLAB\Chapitres
prochains\Programmation_x\sinc2.m')
```

```
W =
        path: 'C:\Documents and Settings\MARTAJ\Mes
documents\MATLAB\Chapitres prochains\Analyse numérique_x'
           m: {'sol_syst_graph.m'}
         mat: {0x1 cell}
         mex: {0x1 cell}
         mdl: {2x1 cell}
           p: {0x1 cell}
     classes: {0x1 cell}
    packages: {0x1 cell}
```

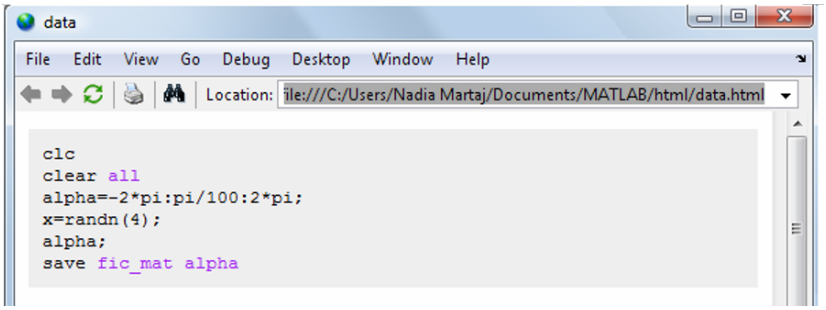
Dans l'espace de travail, nous avons les matrices `x` et `ans` ainsi que la structure `W`.



Nous pouvons sauvegarder ces valeurs dans un fichier et pouvoir récupérer ces variables.

- **`File publish`**

On peut publier un script, comme le fichier de données `data.m`, sous forme html.



Ce fichier est sauvegardé dans un répertoire nommé automatiquement `html`.

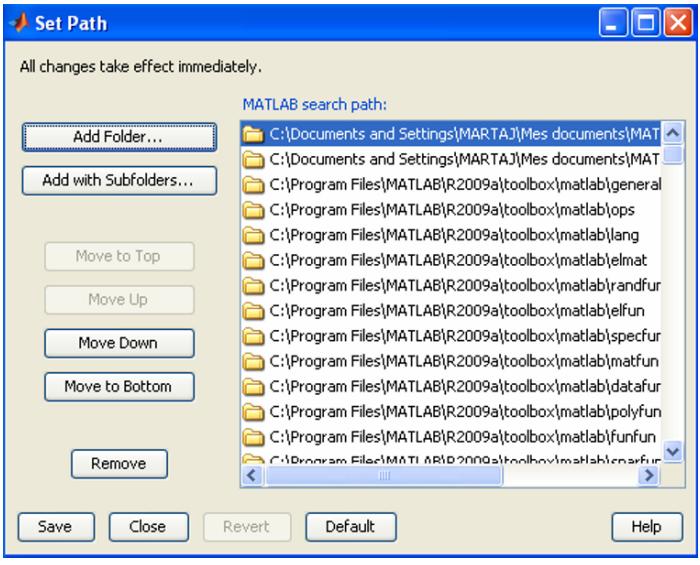
- **`File Set Path`** …

Permet d'ouvrir la boite de dialogue qui permet la gestion des répertoires (ajout avec sans sous-répertoires, suppression de répertoires, etc.).

On peut parcourir la liste des répertoires dans le sens ascendant (`Move to Bottom`) ou descendant (`Move Down`).

Le chemin de recherche commence par le répertoire du haut et finit par le dernier plus bas. On peut ainsi rajouter des répertoires ou en supprimer certains dans le chemin de recherche de MATLAB.

Si on veut ajouter un répertoire, on clique sur le bouton `Add Folder …` et on l'on choisit le répertoire qu'on veut dans la nouvelle boite de dialogue.



- **`File Preferences`** …

Permet de choisir les différentes caractéristiques de MATLAB, SIMULINK, etc., comme le choix du format d'affichage des nombres, la couleur, etc.

- **`Edit Paste to Workspace`**

Enregistre le contenu du fichier dans une structure appelée `A_pastespecial`.

```
>> whos
```

```
  Name                      Size                Bytes   Class
Attributes
  A_pastespecial      3x1                 346  cell
```

Si on demande sa valeur, on obtient :

```
>> A_pastespecial

A_pastespecial =
    'clc, clear all, alpha = -2*pi:pi/100 :2*pi;'
    'x = rand(4); alpha ;'
    'save fic_mat x alpha'
```

- **Edit Find Files**

Permet la recherche de fichiers.

Par ce menu `Edit`, on peut aussi effacer le contenu des différentes fenêtres de MATLAB (`Workspace, etc.`).

- **Text Evaluate Selection (F9)**

Permet d'évaluer les commandes préalablement sélectionnées. On peut remarquer les différentes variables dans la fenêtre `Workspace`.

On peut aussi transformer des lignes de commande en commentaires ou supprimer cette caractéristique, ainsi que réaliser des indentations qu'on peut agrandir ou diminuer.

- **Go**

Permet de déplacer le curseur dans le fichier.

- **Tools**

Permet d'utiliser des outils tel le profiler étudié plus haut, ainsi que `M-Lint` qui peut afficher un rapport des erreurs au fur et à mesure de la programmation.

- **debug**

Permet le déboguage du fichier en installant ou supprimant des points d'arrêt, etc.

- **Window**

Permet la gestion des différentes fenêtres de MATLAB (leur forme, leur disposition, etc.).

- **`Help`**

  Permet d'afficher de l'aide, d'activer ou désactiver sa licence MATLAB, etc.

  Remarque :
  Le menu `Cell` est étudié dans le chapitre correspondant aux cellules, structures et tableaux multidimensionnels.