Introduction à Matlab et Scilab

Matlab, version 6.5 ; Scilab, version 3.0

Ph. Thieullen

2 février 2005

A Les bases de Matlab et Scilab

A.1 Démarrer

Matlab et Scilab sont des logiciels de calcul scientifique très similaires. Matlab est géré par la société Mathworks ; il est plus proféssionnel et payant. Scilab est géré par le Consortium Scilab (INRIA, ENPC) ; il est du domaine publique et gratuit. Ces deux logiciels utilisent un langage de programmation très similaire, ressemblant plutôt à un langage de script qu'à un langage compilé tel que le C ou le Fortran. En général un langage interprété exécute les instructions beaucoup plus lentement qu'un langage compilé. Un langage interprété se justifie par contre par un temps de conception et de débogage extrèmement réduit. Par ailleurs, dans les deux cas, une très large collection de fonctions et de documentations est disponible. Pour l'aide en ligne

```
>>  help nom_fonction   % Matlab
>>  lookfor mot_cle     % Matlab
--> help mot_cle       // Scilab
--> apropos mot_cle    // Scilab
```

Le début d'un commentaire est signalé par `%` sur Matlab, par `//` sur Scilab et s'arrête en fin de ligne. Pour plus d'aide on peut essayer :

```
>>  more on % affichage par page
>>  help topics: % longue liste
>> demo matlab
--> exec(SCI+'/demos/alldems.dem');
```

Pour naviguer dans ces répertoires, on utilise

Matlab Scilab	
`cd`	modification du répertoire courant
`ls`	liste des fichiers et des répertoires
`pwd`	nom du répertoire courant

Par exemple sur un PC en Scilab :

```
--> cd 'c:\Documents and settings...
--> \nom_utilisateur\'
```

On remarque qu'on peut écrire une instruction sur plusieurs lignes, aussi bien sur Matlab que sur Scilab, en terminant la ligne avec

...

Il est possible de travailler directement sur la fenêtre de commande. Mais c'est souvent plus commode d'écrire un script ou des fonctions sur un fichier séparé et d'appeler ce fichier dans la fenêtre de commande. N'importe quel éditeur de fichiers `.txt` convient. Il est possible aussi d'utiliser l'éditeur local du logiciel :

```
--> // Creation d'un script ou
--> // d'une fonction en Scilab
--> scipad nom_script.sce
--> scipad nom_fonction.sci
--> // fichier vide
```

```
--> scipad();
>>  % Creation d'un script ou
>>  % d'une fonction en Matlab
>>  edit nom_script.m
>>  edit nom_fonction.m
>>  % fichier vide
>>  edit
```

Les fichiers script ou fonction ont pour extension (usuelle mais non impérative) `.sce` ou `.sci` en Scilab et `.m` en Matlab. Pour exécuter un script dans la fenêtre de commande, on utilise le nom du fichier qui le contient ; pour Scilab, il faut garder l'extension ; pour Matlab, il ne faut pas la garder :

```
--> //Scilab
--> exec("nom_script.sce");
>>  %Matlab
>>  nom_script
```

Pour utiliser une fonction par la suite, il faut d'abord charger le fichier dans Scilab

```
--> // Sur Scilab :
--> // chargement d'un fonction
--> getf("nom_fonction.sci");
--> // utilisation de la fonction
--> resultat = nom_fonction(args);
>>  % Sur Matlab :
>>  % pas de chargement
>>  % utilisation directe
>>  resultat = nom_fonction(args);
```

Il se peut que la pile des données (l'endroit de la mémoire que le logiciel se réserve) soit insuffisante. Sur Scilab, `stacksize(n)` fixe la taille de la pile et `V = stacksize()` fournit l'état de la pile. Par exemple

```
--> //Scilab
--> n = 2000000
--> stacksize(n)
--> V = stacksize()
--> // V = [2000000, 415463]
```

Sur Matlab, il n'existe pas de fonction équivalente ; il faut manuellement augmenter la taille de la mémoire virtuelle.

Pour sortir de Matlab ou Scilab, on exécute

```
>>  exit  % Matlab
--> exit // scilab
```

A.2 Un super calculateur

On peut utiliser Scilab ou Matlab comme un simple calculateur.

A.2.1 Opérateurs arithmétiques

Matlab Scilab	
$+$	addition
$-$	soustraction
$*$	multiplication
$/$	division
`^`	puissance
()	parenthèse

```
--> -3.5e-01     // ans = -0.35
--> 5.0*10^4     // ans = 50000
--> -2d-6        // ans = -0.000002
>>  -3.5*10^(-1)  % ans = -0.3500
>>  5.0e+4        % ans = 50000
>>  -2d-6      % ans = -2.0000e-006
```

Un retour à la ligne exécute le calcul (à moins que celle-ci ne se termine par trois points `...`) Pour éviter d'afficher à l'écran le résultat du calcul, on termine la ligne par un point virgule `;`.

A.2.2 Constantes prédéfinies

Scilab	Matlab	Constante
`%e`	`exp(1)`	2.7182818
`%eps`	`eps`	précision machine
`%f , %F`	`0`	false (faux)
`%i`	`i , j`	$\sqrt{-1}$
`%inf`	`Inf`	infini
`%nan`	`NaN`	not a number
`%pi`	`pi`	3.1415927
	`realmax`	plus grand réel
	`realmin`	plus petit réel
`%s`		indéterminée
`%t , %T`	`1`	true (vrai)

```
--> cos(%pi) // ans = -1.
--> exp(%i*%pi/3)
--> // ans =  0.5 + 0.8660254i
--> %eps // %eps  = 2.220D-16
>>  1/2 +i*sqrt(3)/2
>>  % ans = 0.5000 + 0.8660i
>>  0/0 % ans = NaN
>>  eps % ans = 2.2204e-016
```

Dans des calculs arithmétiques, sur Scilab, les constantes booléennes `%F` et `%T` se comportent comme les scalaires 0 et 1.

```
--> x = 1.2;
--> signe = 0+(x>=0)-(x<=0)
--> //signe = 1
```

A.2.3 Types des données

Les types de données les plus importants sont : les types scalaires (entier, réels ou complexes), les types caractères. Pour connaître le type de la donnée, suivant Matlab ou Scilab, on écrira

Matlab	
`var`	variable
`class(var)`	type de la variable
Scilab	
`var`	variable
`typeof(var)`	type de la variable

On récapitule dans les deux tableaux suivant la liste exaustive des types renvoyés.

Scilab	**variable**
`double`	précision double
`logical`	booléen
`char`	caractère
`cell`	cellule
`function_handle`	pointeur fonction
`int8`	entier signé 8 bits
`uint8`	entier non signé
`int16`	entier signé 16 bits
`uint16`	entier non signé
`int32`	entier signé 32 bits
`uint32`	entier non signé
`<class_name>`	objet java
`<java_class>`	nom de classe java

Scilab	**variable**
`constant`	réelle ou complexe
`polynomial`	polynomiale
`function`	fonction
`handle`	pointeur
`string`	chaîne de caractères
`boolean`	booléenne
`list`	liste
`rational`	fraction rationnelle
`state-space`	modèle alébrique
`sparse`	mat. réelle creuse
`boolean sparse`	mat. bool. creuse

Scilab Matlab	**Type entier**
`int8`	entier signé sur 1 octet
`uint8`	entier non signé sur 1 octet
`int16`	entier signé sur 2 octets
`uint16`	entier non signé sur 2 octets
`int32`	entier signé sur 4 octets
`uint32`	entier non signé sur 4 octets
`int64`	entier signé sur 8 octets
`uint64`	entier non signé sur 8 octets

Matlab Scilab	
`single`	précision simple
`double`	précision double

A.2.4 Fonctions standard

Matlab Scilab	
`acos`	fonction inverse à cosinus
`asin`	fonction inverse à sinus
`atan`	fonction inverse à tangente
`cos`	fonction cosinus
`exp`	fonction exponentielle
`log`	fonction logarithme
`sin`	fonction sinus
`sqrt`	fonction racine carrée
`tan`	fonction tangente

```
--> sqrt(-1)   // ans = i
--> cos(%pi/2) // ans  = 6.123D-17
>>  sqrt(-1)    % ans = 0 + 1.0000i
>>  sin(pi)     % ans = 1.2246e-016
```

Matlab Scilab	
`ceil`	partie entière supérieure
`fix`	`y=sign(x).floor(abs(x))`
`floor`	partie entière inférieure
`round`	plus proche entier
`sign`	fonction signe (-1,0,1)

```
--> ceil(-1.9)    // ans = -1.
--> fix(-%e)      // ans = -2.
--> round(0.4999) // ans = 0.
>>  floor(-1.9)    % ans = -2
>>  round(1/2)     % ans = 1
>>  sign(-pi)      % ans = -1
```

`abs`	module
`angle`	Matlab : argument (radian)
`conj`	conjugué d'un complexe
`imag`	partie imaginaire
`phasemag`	Scilab : argument (degré)
`real`	partie réele

```
--> ii = exp(%i*%pi/2)
--> // ii = 6.123D-17 + i
--> abs(ii) // ans = 1.
--> [phase,gain] = phasemag(ii)
--> // gain = 0. phase = 90.
--> atan(imag(ii),real(ii))
--> // ans  = 1.5707963
>>  ii = exp(i*pi/2)
>>  % ii = 0.0000 + 1.0000i
>>  angle(ii) % ans = 1.5708
```

`mod`	(M) reste euclidien (`floor`)
`pmodulo`	(S) reste euclidien (`floor`)
`rem`	(M) reste euclidien (`fix`)
`modulo`	(S) reste euclidien (`fix`)

(M) pour Matlab, (S) pour Scilab

A.3 Variables

On vient d'utiliser une variable `ii`. Les règles de construction d'un nom de variable diffèrent entre Matlab et Scilab.

A.3.1 Règles de Matlab

Un nom de variable est constitué de chiffres, de lettres et de caractères spéciaux

`12...9`	`abc...xyz`	`_`

Un nom doit commencer par une lettre. Seuls les `namelengthmax = 63` premiers caratères sont pris en compte. Majuscules et minuscules sont prises en compte.

```
>> AaA, A_a_A, A1a1A   % valide
>> _abc, 123           % invalide
```

A.3.2 Règles de Scilab

un nom de variable est constitué de chiffres, de lettres et de caractères spéciaux

`12...9`	`abc...xyz`	`% _ # ! $ ?`

Un nom ne peut pas commencer par un chiffre. Le caractère `%` ne peut apparaître qu'en première position. Seuls les 24 premiers caratères sont pris en compte. Majuscules et minuscules sont prises en compte.

```
--> %eps, un_nom, #123  // valide
--> 1abc, A%A, .C       // invalide
```

A.3.3 Quelques commentaires

Plusieurs expressions peuvent être écrites sur une même ligne à condition qu'elles soient séparées par une virgule (le résultat de l'expression est affiché), ou bien par un point virgule (le résultat n'est pas affiché).

La variable `ans` contient le résultat du dernier calcul non affecté. Toutes les variables d'une session, y compris celles d'un script qu'on exécute, sont globales et sont donc conservées en mémoire.

A.4 Opérateurs logiques

Matlab Scilab	
`==`	égal à
`~=`	différent de
`<`	inférieur à
`>`	supérieur à
`<=`	inférieur ou égal à
`>=`	supérieur ou égal à
`~`	non
`&`	et
`|`	ou

Par exemple `(1.5<2)&(1.5>=1)` donne `ans = 1` pour Matlab et `ans = T` pour Scilab.

A.5 Vecteurs et matrices

Une des grandes forces de Matlab et Scilab est de pouvoir travailler très rapidement sur des matrices. A partir de maintenant, l'invite `-->` de Scilab ou `>>` de Matlab n'est plus reproduite.

Un commentaire sera indiqué lorsque un logiciel possède une construction propre.

A.5.1 Construction

Un vecteur ligne, colonne ou une matrice se construit en insérant ses entrées entre deux crochets `[]`. La virgule sert à séparer les entrées d'une ligne, le point-virgule, les entrées d'une colonne. Par exemple

```
A = [1,2,3,4;5,6,7,8]
B = [1,2,3], C = [1;2;3]
D = [1 2 3 4
     5 6 7 8]
```

donne

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$
$$C = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}.$$

On peut aussi construire des matrices par blocs par concaténation. Par exemple

```
A = [1,2;3,4]; B = [5;6];
C = [7,8]; D = 9; E = [A,B;C,D]
```

donne

$$E = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \begin{bmatrix} 5 \\ 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 \end{bmatrix} & \begin{bmatrix} 9 \end{bmatrix} \end{bmatrix}$$

A.5.2 Notation "deux points"

Il existe une syntaxe très pratique pour construire un vecteur ligne. La notation deux points :

Matlab Scilab
`val_initiale :pas :val_finale`
`val_initiale :val_finale`

Dans la deuxième construction, le pas est sous-entendu égal à 1. Par exemple

```
1:10, 1:1:10, 1:2:10, 1:-2:-10
0:0.05:0.33
```

donne respectivement

$$1:10 = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$
$$1:1:10 = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$$
$$1:2:10 = 1\ 3\ 5\ 7\ 9$$
$$1:-2:10 = 1\ \text{-1}\ \text{-3}\ \text{-5}\ \text{-7}\ \text{-9}$$
$$0:0.05:0.33 = 0\ 0.05\ 0.1\ 0.15\ 0.2\ 0.25\ 0.3$$

A.5.3 Extraction

Un vecteur (colonne ou ligne) est en fait une matrice d'un type particulier ($m \times 1$ ou $1 \times n$). On accède donc aux données d'un vecteur à partir de l'indice 1 : $V(1)$ désigne la première coordonnée de V.

Pour une matrice A de dimension $m \times n$, $A(i,j)$ désigne l'entrée à la ième ligne et à la jème colonne. Par exemple

```
A = [0:2:16];
A(1), A(4), A(9)
B = [1,2,3;4,5,6];
B(1,1), B(2,3)
```

donne respectivement

$$ans \quad = \quad 0, \quad 6, \quad 16, \quad 1, \quad 6.$$

On peut aussi extraire des sous-matrices $A(i,j)$ en prenant cette fois ci pour i et j des vecteurs (ligne ou colonne) d'indices. Si i ou j est égal au caractère (deux points) :, il est sous entendu que toutes les lignes ou toutes colonnes sont retenues. Par exemple

```
A = [1:5;5:-1:1;[0,0,0,0,0]]
B = A([1,3],1:2:5)
C = A(:,[2;4]), D = A(1,:)
```

donne respectivement

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 3 & 5 \\ 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 2 & 4 \\ 4 & 2 \\ 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

Pour une matrice A, $A(:)$ désigne le vecteur colonne obtenu en mettant bout-à-bout toutes les colonnes de A. Modifier le contenu d'une matrice se fait de la même manière. Dans l'exemple suivant, la matrice A prend les deux valeurs :

```
A = [1:4;2:5;3:6;4:7];
A(:,[2,4])=[]; B = A
A(:,2)=[4;3;2;1]; C = A
```

$$B = \begin{bmatrix} 1 & 3 \\ 2 & 4 \\ 3 & 5 \\ 4 & 6 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 4 \\ 2 & 3 \\ 3 & 2 \\ 4 & 1 \end{bmatrix}.$$

On récapitule dans le tableau suivant les différentes constructions :

• Pour les vecteurs (ligne ou colonne)

Matlab Scilab	
`V`	vecteur de dim. m
`ii`	indice : $1 \leq ii \leq m$
`V(ii)`	iième composante de V
`II`	$II = [i_1, \cdots, i_r]$, $1 \leq i_k \leq m$
`V(II)`	$[V(i_1), V(i_2), \cdots, V(i_r)]$

• Pour les matrices

Matlab Scilab	
`A`	matrice de dim. $m \times n$
`ii,jj`	$1 \leq ii \leq m$, $1 \leq jj \leq n$
`A(ii,jj)`	entrée (ii, jj) de A
`II`	$II = [i_1, \cdots, i_p]$, $i_k \leq m$
`JJ`	$JJ = [j_1, \cdots, j_q]$, $j_k \leq n$
`A(II,JJ)`	matrice de dim. $p \times q$
`A( :,JJ)`	matrice de dim. $m \times q$
`A(II, :)`	matrice de dim. $p \times n$
`A( :)`	vect. colonne de dim. mn

Les vecteurs lignes II et JJ sont composés d'indices de ligne ou de colonne de la matrice initiale A ; il n'est pas interdit de répéter ces indices. Par exemple

```
A = [0,1,2;1,2,0;2,0,1]
B = A([1,1],[3,2,1])
```

donne

$$A = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

On peut aussi extraire des données d'une matrice en utilisant comme indice des variables booléennes. Si A est une matrice réelle (ou complexe) et si B est une matrice booléenne de même dimension, $A(B)$ retourne un vecteur des entrées $A(i, j)$ pour lesquelles $B(i, j)$ est vraie. Sur Scilab, les entées de B sont du type `%T` ou `%F`. Sur Matlab, il faut convertir une matrice de `0` ou de `1` au moyen de la fonction `logical`. Pour des vecteurs, `V(W)` retourne un vecteur de même type. Pour des matrices, `A(B)` retourne toujours un vecteur colonne.

Matlab Scilab	
`V`	vect. de dim m
`W`	vect booleèn de dim m
`V(W)`	$V(i)$ si $W(i) =$ `vrai`
`A`	matrice $m \times n$
`B`	matrice booleènne $m \times n$
`A(B)`	$A(i, j)$ si $B(i, j) =$ `vrai`

Par exemple

```
%Matlab
V = (1:10)
W = V(logical([0,0,0,0,...
    1,1,1,1,1,1]))
//Scilab
V = (1:10)
W = V([%F,%F,%F,%F,...
    %T,%T,%T,%T,%T,%T])
```

donne

$$W = [5, 6, 7, 8, 9, 10]$$

```
%Matlab
A = [1+i,1,1-i;1,0,-1;-i,i,0]
B = logical([1,0,1;0,1,0;...
    1,0,1])
C = A(B)
\\Scilab
A = [1+%i,1,1-%i;1,0,-1;-%i,%i,0]
B = A([%T,%F,%T;%F,%T,%F;...
    %T,%F,%T])
```

donne

$$A = \begin{bmatrix} 1+i, & 1 & 1-i \\ 1 & 0 & -1 \\ -i & i & 0 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$A(B) = \begin{bmatrix} 1+i & -i & 0 & 1-i & 0 \end{bmatrix}^{\mathrm{T}}$$

Les colonnes sont parcourus les unes après les autres de haut en bas. On retiendra la fonction spécifique à Matlab

Matlab	
`A`	matrice scalaire
`B=logical(A)`	B booléenne

où $B(i,j) =$ TRUE si et seulement si $A(i,i) \neq 0$. On retiendra aussi la fonction spécifique à Scilab

Scilab	
`A`	matrice booléenne
`B=bool2s(A)`	matrice de 0 et de 1

transformant une matrice booléenne en une matrice de 0 et de 1 où `1` $\simeq$ `%T` et `0` $\simeq$ `%F`.

A.5.4 Dimensions

Pour calculer la dimension d'un vecteur ligne ou colonne, on utilise la fonction `length`. Pour calculer les dimensions d'une matrice, on utilise la fonction `size` qui retourne un vecteur ligne

Matlab	**Scilab**
`V`	vecteur de dim. m
`length(V)`	le réel m
`A`	matrice de dim. $m \times n$
`size(A)`	le vecteur `[m,n]`
`size(A,1)`	le réel m
`size(A,2)`	le réel n

Par exemple

```
A = [1:2:9;9:-2:1];V = A(2,:);
dim_A = size(A),dim_V = length(V)
nb_lignes = size(A,1),
nb_colonnes = size(A,2)
```

donne

```
dim_A=[2,5], dim_V=5
nb_lignes=2, nb_colonnes=5
```

Scilab possède ici une construction particulière :

Scilab	
`A`	matrice de dim. $m \times n$
`size(A,"r")`	nombre de lignes m
`size(A,"c")`	nombre de colonnes n
`size(A,"*")`	nombre total $m * n$

A.5.5 Opérations matricielles

Le tableau ci-dessous résume toutes les opérations matricielles avancées que ces deux logiciels acceptent :

Matlab Scilab	
`A,B`	matrices de dim. $m \times n$
`C`	matrice de dim $n \times p$
`s`	scalaire
`A'`	transposé + conjgué
`A.'`	transposé uniquement
`s*A`	produit par un scalaire
`A/s`	division par un scalaire
`A+B`	addition matricielle
`A*C`	produit matricielle
`A^s`	puissance d'une matrice
`A.*B`	produit terme à terme
`A./B`	division terme à terme
`s./A`	division terme à terme
`A.^B`	puissance terme à terme
`s.^A`	puissance terme à terme
`A.^s`	puissance terme à terme

Par exemple, le produit scalaire euclidien de deux vecteurs lignes `U` et `V` s'écrit très simplement par `U*V'` ; ici `V'` représente le vecteur colonne transposé de `U` :

```
U = [1,2,3,4]; V = [1,-1,3,-2];
a = U.*V, b = sum(a), c=  U*V'
```

et donne

$$a = [1, -2, 9, -8], \quad b = 0, \quad c = 0$$

Dans l'exemple suivant, remarquer bien la différence d'utilisation des constructions `V.^2` et `A^2`

```
V = 1:10, W = V.^2
A = [0,0,0,1;1,0,0,0;...
     0,1,0,0;0,0,1,0]
B = A^4
```

donne

$$V = [1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10],$$
$$W = [1,\ 4,\ 9,\ 16,\ 25,\ 36,\ 49,\ 64,\ 81,\ 100],$$
$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Les opérateurs logiques s'appliquent aussi aux matrices. Par exemple, si A et B sont deux matrices de même dimension, $A < B$ est une matrice booléenne (d'entrées 0,1 pour Matlab ou `%F`,`%T` pour Scilab) ou chaque entrée (i, j) est le résultat bouléen $A(i, j) < B(i, j)$; si s est un scalaire $s < A$ est aussi une matrice booléenne d'entrée (i, j) égal au résultat booléen $s < A(i, j)$.

Matlab Scilab	
`A,B`	matrice de dim $m \times n$
`s`	scalaire
`A==B`	$A(i,j) == B(i,j$
`A~=B`	$A(i,j) \sim= B(i,j)$
`A<B`	$A(i,j) < B(i,j)$
`A>B`	$A(i,j) > B(i,j)$
`A<=B`	$A(i,j) <= B(i,j)$
`A>=B`	$A(i,j) >= B(i,j)$
`A`	$\sim A(i,j)$
`A&B`	$A(i,j\&B(i,j)$
`A|B`	$A(i,j)\|B(i,j)$

Par exemple

```
V = 1:10; W = V(V>=5)
```

donne `W=[ 5, 6, 7, 8, 9, 10]`.

Les fonctions suivantes permettent de savoir si un vecteur ou matrice possède au moins une entrée non nulle, `any` ou `or`, ou toutes les entrées non nulles, `all` ou `and`. Matlab et Scilab diffèrent ici :

Matlab	
`s`	scalaire
`V`	vecteur ligne
`W`	vecteur colonne
`A`	matrice $m \times n$
`s = any(V)`	$s = \sup_i \text{sign}\|V(i)\|$
`V = any(A,1)`	$V_j = \sup_i \text{sign}\|A(i,j)\|$
`W = any(A,2)`	$W_i = \sup_j \text{sign}\|A(i,j)\|$
`s = all(V)`	$s = \inf_i \text{sign}\|V(i\|$
`V = all(A,1)`	idem que pour `any`
`V = all(A,2)`	idem que pour `any`

Scilab	
`V,W`	vecteur bool. ou scal.
`A`	matrice bool. ou scal.
`s = or(V)`	`%T` si $V \neq 0$
`V = or(A,1)`	`%T` si col.$(j) \neq 0$
`W = or(A,2)`	`%T` si lig.$(i) \neq 0$
`s = and(V)`	`%T` si tous $V(i) \neq 0$
`V = and(A,1)`	`%T` si tous $A(:, j) \neq 0$
`W = and(A,2)`	`%T` si tous $A(i, :) \neq 0$

Par exemple

```
//Scilab
A = [0,0,1.5;0,0.5,-1]
B = bool2s(A~=0)
V = or(A,1), W = and(A,1)
%Matlab
A = [0,0,1.5;0,0.5,-1]
B = (A~=0)
V = any(A,1), W = all(A,1)
```

donne

$$A = \begin{bmatrix} 0 & 0 & 1.5 \\ 0 & 0.5 & -1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

pour la version Scilab

$$V = [\mathtt{F}, \mathtt{T}, \mathtt{T}] \quad W = [\mathtt{F}, \mathtt{F}, \mathtt{T}]$$

et pour la version Matlab

$$V = [0, 1, 1] \quad W = [0, 0, 1].$$

On termine cette partie par un exemple un peu plus développé. Représenter une fonction $y = f(x)$ revient à se donner un vecteur d'absisses $x = [x_1, x_2, \cdots, x_n]$ et le vecteur d'ordonnées correspondant $y = [f(x_1), f(x_2), \cdots, f(x_n)]$. Par exemple pour définir numériquement la fonction sinus cardinal ($f(x) = \sin(x)/x$ si $x \neq 0$, $f(0) = 1$) sur l'intervalle $[-5\pi, 5\pi]$ au pas de 0.001, on écrit le code

```
//Scilab
x = 5*%pi*(-1:0.001:1);
delta = (x==0);
y = (sin(x) + delta)./(x + delta);
plot2d(x,y) //plot(x,y) pour Matlab
```

`delta` est un vecteur ligne booléen de même longueur que `x` valant 0 (ou `%F`) partout sauf à l'indice i telque `x(i)=0`. On aurait pu écrire plus simplement `sin(x)./x` mais on aurait obrenu une erreur à cet indice i. Le vecteur `delta` sert donc à corriger ce problème. La fonction `plot2d` sera examinée plus tard.

A.5.6 Matrices spéciales

Matlab Scilab	
`zeros(m,n)`	matrice $m \times n$ de 0
`ones(m,n)`	matrice $m \times n$ de 1
`eye(m,n)`	matrice identité $m \times n$

Par exemple

```
A = [1,2,3;4,5,6]
[nb_lignes, nb_colonnes] = size(A)
B = eye(nb_lignes, nb_colonnes)
// version specifique Scilab
B = eye(A), C = eye(size(A))
% version specifique Matlab
% eye(A) provoque une erreur
B = eye(size(A))
```

donne

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix},$$
$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

La fonction `diag` permet, aussi bien d'extraire la kième diagonale d'une matrice (sous forme d'un vecteur colonne), que de construire des matrices à partir de ces diagonales.

Matlab Scilab	
`A`	matrice $m \times n$
`k`	entier > 0 ou < 0
`V`	vecteur colonne
`V = diag(A)`	diagonale principale
`V = diag(A,k)`	kième diagonale
`A = diag(V)`	$m = n = \dim(V)$
`A = diag(V,k)`	$m = n = \dim(V) + k$

Dans la construction `A = diag(V,k)`, la matrice obtenue est carrée de dimension $\dim(V) + k$ et a toutes ses entrées nulles sauf la kième diagonale égale au vecteur (colonne) V. Par exemple

```
A = [1,2,3;4,5,6];
V = diag(A), W = diag(A,1)
B = diag([2;2;2])+diag([-1;-1],1)...
    + diag([-1;-1],-1)
```

donne

$$V = \begin{bmatrix} 1 \\ 5 \end{bmatrix}, \; W = \begin{bmatrix} 2 \\ 6 \end{bmatrix}, \; B = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}.$$

Sur le même principe que la fonction `diag`, on dispose aussi de `triu` et de `tril` qui extraient d'une matrice ses parties triangulaires supérieures et inférieures.

Matlab Scilab	
`A`	matrice $m \times n$
`k`	entier > 0 ou < 0
`triu(A)`	triangulaire supérieure
`triu(A,k)`	idem depuis la diag. k
`tril(A)`	triangulaire inférieure
`tril(A,k)`	idem depuis la diag. k

Par exemple

```
A = [1,2,3;4,5,6];
B = triu(A,-1), C = tril(A,-1)
```

donne

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

A.5.7 Opérations avancées

Les fonctions suivantes agissent aussi bien sur des vecteurs que sur des matrices et évitent d'utiliser des boucles de programmation (très couteux en temps pour des langages interprétés).

Matlab Scilab	
`A`	matrice $m \times n$
`V`	vect. ligne $1 \times n$
`W`	vect. colonne $m \times 1$
`m`	scalaire
`m = sum(V)`	$m = \sum_k V(k)$
`V = sum(A,1)`	somme par colonne
`W = sum(A,2)`	somme par ligne
`m = prod(V)`	$m = \prod_k V(k)$
`V = prod(A,1)`	idem pour produit
`W = prod(A,2)`	idem pour produit
`cumsum(V)`	sommes cumulées
`cumsum(A,1)`	somme par colonne
`cumsum(A,2)`	somme par ligne
`cumprod(V)`	produits cummulés
`cumprod(A,1)`	idem pour produit
`cimprod(A,2)`	idem pour produit

```
A = [1,2,3,4;2,3,4,5;6,7,8,9]
B = sum(A,1),C = sum(A,2)
D = cumsum(A,1), E = cumsum(A,2)
F = prod(A,1), G = cumprod(A,2)
```

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 9 & 12 & 15 & 18 \end{bmatrix}$$

$$C = \begin{bmatrix} 10 \\ 14 \\ 30 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 5 & 7 & 9 \\ 9 & 12 & 15 & 18 \end{bmatrix}$$

$$E = \begin{bmatrix} 1 & 3 & 6 & 10 \\ 2 & 5 & 9 & 14 \\ 6 & 13 & 21 & 30 \end{bmatrix}$$

$$F = \begin{bmatrix} 12 & 42 & 96 & 180 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 2 & 6 & 24 \\ 2 & 6 & 24 & 120 \\ 6 & 42 & 336 & 3024 \end{bmatrix}$$

Toujours pour éviter d'écrire des petits programmes élémentaires mais très lents, on dispose aussi de

Matlab Scilab	
`V`	vecteurs lig. ou col.
`m,k`	scalaires
`m = min(V)`	$m = \min_i V(i)$
`[m,k] = min(V)`	$m = V(k)$
`m = max(V)`	$m = \max_i V(i)$
`[m,k] = max(V)`	$m = V(k)$

Les fonctions `min` et `max` agissent aussi sur des matrices mais Matlab et Scilab diffèrent ici sur la syntaxe. Dans ce qui suit

`A` : matrice $m \times n$
`M1,K1` : vecteurs lignes
`M2,K2` : vecteurs colonnes
`m,k` : scalaires

$M1(j) = \min_i A(i,j)$ désigne le minimum de chaque colonne, $M2(i) = \min_j A(i,j)$, le minimum de chaque ligne, $K1(j)$, l'indice de ligne minimisant le vecteur colonne j, $K2(i)$, l'indice de colonne minimisant le vecteur ligne i.

Matlab
`m = min(V)`
`[m,k] = min(V)`
`M1 = min(A,1)`
`M2 = min(A,2)`
`[M1,K1] = min(A,[],1)`
`[M2,K2] = min(A,[],2)`
idem pour max

Scilab
`m = min(V)`
`[m,k] = min(V)`
`M1 = min(A,1)`
`M2 = min(A,2)`
`[M1,K1] = min(A,'r')`
`[M2,K2] = min(A,'c')`
idem pour max

Par exemple

```
%Matlab
A = [1,2,3;3,2,1]
[M1,K1] = min(A,[],1)
[M2,K2] = min(A,[],2)

//Scilab
A = [1,2,3;3,2,1]
[M1,K1] = min(A,'r')
[M2,K2] = min(A,'c')
```

donne dans les deux cas

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix},\ M2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix},\ K2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$
$$M1 = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix},\ K1 = \begin{bmatrix} 1 & 1 & 2 \end{bmatrix}.$$

Dans l'exemple suivant, on cherche à minimiser la fonction $f(x) = x^4 - x^2$ sur l'intervalle $-10 \leq x \leq 10$. Théoriquement, le minimum vaut $m = -1/4$ et les points qui réalisent ce minimum sont $x_{\min} = \pm\sqrt{1/2}$, c'est-à-dire $x_{\min} = \pm 0.7071068$. Pour une manère (peu élégante) de trouver une solution, le code suivant :

```
x = -10:0.0001:10; y = x.^4-x.^2;
m = min(y), [m,k] = min(y);
x_min = -10+0.0001*k, sqrt(1/2)
```

donne

$$m = -0.25 \quad x_{\min} = -0.707$$
$$ans = 0.7071068.$$

Enfin on dispose de la fonction très commode `find` qui permet d'extraire les entrées d'un vecteur V ou d'une matrice A qui ne sont pas nulles. Par exemple, si V est un vecteur, `find(V)` retourne un vecteur d'indices $i_1, i_2, \cdots$ (toujours ligne en Scilab et de même type que V en Matlab) tel que $V(i_1) \neq 0$, $V(i_2) \neq 0$,...et $V(i) = 0$ partout ailleurs. Si A est une matrice, `find(A)` retourne deux vecteurs $(i_1, i_2, \cdots)$ et $(j_1, j_2, \cdots)$ tels que $A(i_1, j_1) \neq 0$, $A(i_2, j_2) \neq 0$,...et $A(i, j) = 0$ partout ailleurs.

Matlab Scilab	
`V`	vect. lig. ou col.
`A`	matrice reelle
`II,JJ`	vect. lig. ou col.
`II = find(V)`	$V(i) \neq 0$
`[II,JJ] = find(A)`	$A(i,j) \neq 0$

Par exemple

```
A = 2*eye(3,3) - diag([1,1],1)...
    - diag([1,1],-1)
[II,JJ] = find(A)
```

donne

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$
$$JJ = \texttt{[1. 1. 2. 2. 2. 3. 3.]},$$
$$II = \texttt{[1. 2. 1. 2. 3. 2. 3.]}$$

(et le même résultat sous forme de vecteur colonne sous Matlab).

Bienque un peu spécialisée aux problèmes statistique, on trouve dans la configuration de base des fonctions, appelées générateurs aléatoires de nombres, qui simulent un échantillon d'une loi donnée. Les deux gńérateurs donnée en standard sont : la loi uniforme sur $[0, 1]$, la loi normale sur $\mathbb{R}$, centrée réduite. Là encore Matlab et Scilab diffèrent.

Matlab	
`k,m,n`	entiers
`A`	matrice $m \times n$
`A = rand(m,n)`	distrib. uniforme
`A = randn(m,n)`	distrib. normale
`rand('state',k)`	initialise l'amorce

Pour modifier l'amorce à chaque appelle de `rand` ou `randn`, on peut executer

```
rand('state',100*sum(clock))
randn('state',100*sum(clock))
```

Scilab	
`m,n`	entiers
`A = rand(m,n,'uniform')`	uniforme
`A = rand(m,n,'normal')`	normale
`rand('seed',k)`	amorce k

Par exemple

```
%Matlab
V = rand(1,5)
rand('seed')
//Scilab
W = rand(1,5,'normal')
```

donne

```
V = [0.6154, 0.7919,...
     0.9218, 0.7382, 0.1763]
W = [0.0698768, -2.8759125,...
    -1.3772844,  0.7915156,...
    -0.1728369]
```

Toujours en relation avec des questions statistiques, les deux logiciels proposent des fonctions de tri, de moyenne et d'écart-type.

Matlab et Scilab diffèrent pour la fonction de tri `sort` dans l'ordre des données triées. Pour Matlab les données sont triées par ordre croissant, pour Scilab, par ordre décroissant. Pour une matrice `C`,

```
[A,B] = sort(C,1)
```

donne en sortie une matrice `A` triée suivant les lignes pour chaque colonne et une matrice de vecteurs colonnes de permutation qui assure l'identité

```
C(B( :,j),j) = A( :,j)
```

Matlab et Scilab	
`A, B, C`	vecteurs
`[A,B]=sort(C)`	A mat. triée
Matlab	
`A, B, C`	mat. $m \times n$
`[A,B]=sort(C,1)`	trie les lig.
`[A,B]=sort(C,2)`	trie les col.
Scilab	
`A, B, C`	mat. $m \times n$
`[A,B]=sort(C,'r')`	trie les lig.
`[A,B]=sort(C,'c')`	trie les col.

```
//Scilab
C = ceil(10*rand(1,5))
[A,B] = sort(C)
//C = [1, 7, 3, 4, 9]
//B = [5, 2, 4, 3, 1]
//A = [9, 7, 4, 3, 1]
%Matlab
C = ceil(10*rand(3,5))
[A,B] = sort(C,1)
```

$$C = \begin{bmatrix} 9 & 4 & 8 & 2 & 4 \\ 1 & 9 & 5 & 2 & 6 \\ 7 & 6 & 4 & 7 & 2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 4 & 4 & 2 & 2 \\ 7 & 6 & 5 & 2 & 4 \\ 9 & 9 & 8 & 7 & 6 \end{bmatrix}$$

$$B = \begin{bmatrix} 2 & 1 & 3 & 1 & 3 \\ 3 & 3 & 2 & 2 & 1 \\ 1 & 2 & 1 & 3 & 2 \end{bmatrix}$$

Matlab Scilab	
`X`	matrice $m \times n$
`A`	vect. lig. n
`B`	vect. col. m
`A=mean(X,1)`	moyenne par col.
`B=mean(X,2)`	moyenne par lig.
`A=median(X,1)`	médiane par col.
`B=median(X,2)`	médiane par lig.

Là encore, dans le cas de Scilab, on peut remplacer `A=mean(X,1)` par `A=mean(X,'r')` et `B=mean(X,2)` par `B=mean(X,'c')`.

```
xx = rand(1,1000,'uniform');
mean(xx,2)
//ans = 0.4957745
```

Pour le calcul de l'écart-type σ Matlab et Scilab utilisent des fonctions différentes. Néanmoins, dans les deux cas, le coefficient de normalisation et $N-1$ où N est la taille des données.

Ecart-type σ	
`X`	mat. $m \times n$
`A`	vect. lig. n
`B`	vect. col. m
Matlab	
`A=std(X,1)`	σ / col.
`B=std(X,2)`	σ / lig.
Scilab	
`A=st_deviation(X,1)`	σ / col.
`B=st_deviation(X,2)`	σ / lig.

Dans l'exemple suivant, on modélise une loi binomiale de paramètres $p = \frac{1}{3}$ et $n = 4$. Les valeurs `PP0` ...`PP4` représentent les fréquences empiriques d'obtenir 0,1...,4. Les valeurs `pp0` ...`pp4` représentent les probabilités théoriques.

```
//Scilab
pp = 1/3;
xx = bool2s(...
     rand(4,10000,'uniform')>1-pp);
XX = sum(xx,'r');
PP0 = mean(bool2s(XX==0))
PP1 = mean(bool2s(XX==1))
PP2 = mean(bool2s(XX==2))
PP3 = mean(bool2s(XX==3))
PP4 = mean(bool2s(XX==4))
pp0 = (2/3)^4
pp1 = 4*(1/3)*(2/3)^3
pp2 = 6*(1/3)^2*(2/3)^2
pp3 = 4*(1/3)^3*(2/3)
pp4 = (1/3)^4
```

Fréquences empiriques				
$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
0.198	0.396	0.295	0.097	0.014

Probabilités théoriques				
$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
0.198	0.395	0.296	0.099	0.012

A.6 Chaînes de caractères

Une chaîne de caractères est crée par la construction suivante :

```
%Matlab
a1 = 'Allemagne,', b1 = 'Belgique,',
c1 = ['C','h','i','l','i']
pays = [a1,' ',b1,' ',c1]
% pays='Allemagne, Belgique, Chili'
//Scilab
a2 = 'Berlin,', b2 = "Bruxelle,"
capitales = a2 + ' ' + b2
//  capitales = 'Berlin, Bruxelle,'
```

En fait, une chaîne de caractères pour Matlab est un tableau $1 \times n$ de caractères ; la concaténation se fait donc comme pour un tableau de scalaires. Pour Scilab, c'est un type de données comme les scalaires. Les opérations suivantes peuvent être réalisées

Matlab	
`A, B`	chaîne de caract.
`strcmp(A,B)`	$ans = 1 \Leftrightarrow A = B$
`isletter(A)`	ans vect. bool. 0/1
`isspace(A)`	ans vect. bool. 0/1

Par exemple

```
strcmp('abc','abc') % ans = 1
'abcd'=='aefd' % ans = [1,0,0,1]
```

Scilab	
grep(A,B)	recherche de motifs
part(A,B)	extraction
strcat(A,B)	concatenation
strindex(A,B)	recherche de motifs
strsubst(A,B,C)	substitution
tokens(A,B)	recherche des tokens

Par exemple :

```
//utilisation de "grep"
aa = ['abc','bcd','cde','dea'];
bb = grep(aa,['a','b']);
//bb = [1,2,4]

//utilisation de "part"
aa = ['abcd','bcda','cdab','dabc'];
bb = part(aa,1:3);
//bb = ['abc','bcd','cda','dab']

//utilisation de "strcat"
aa = ['a','b','c','d'];
bb = strcat(aa);
cc = strcat(aa,'-');
//bb = 'abcd'; cc = 'a-b-c-d'

//utilisation de "strindex"
aa = "12341234512345612";
bb = strindex(aa,"12")
//bb = [1,5,10,16];

//utilisation de "strsubst"
aa = ['a1','a2';'a3','a4'];
bb = strsubst(aa,'a','b');
//bb = [4b1','b2';'b3','b4']

//utilisation de "tokens"
aa = "a aa aaa aaaa";
bb = tokens(aa)
//bb = ['a';'aa';'aaa';'aaaa']
aa = "a+aa-aaa*aaaa";
bb = tokens(aa,['+','-','*'])
//bb = ['a';'aa';'aaa';'aaaa']
```

A.7 Graphiques en 2D

Matlab et Scilab ont deux approches très différentes pour représenter des graphiques. Il existe néanmoins une approche minimaliste commune aux deux logiciels. Dans ce qui suit `X` et `Y` sont des vecteurs ou des matrices.

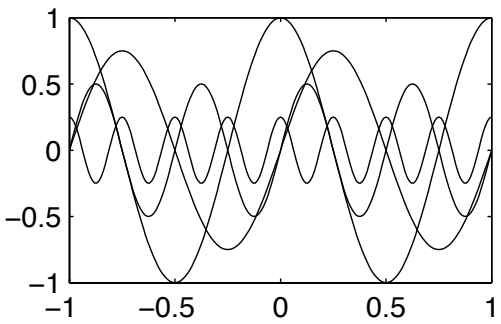
Matlab	
`plot(X,Y)`	Y versus X
Scilab	
`plot2d(X,Y)`	Y versus X
Matlab Scilab	
`clf`	effacement
`subplot(m,n,k)`	sous graphique

Il y a plusieurs cas d'utilisation :

- X et Y sont deux vecteurs de même longueur m, le logiciel trace alors les points $(X(i), Y(i))$, pour $i = 1, 2, \cdots, m$,
- X est un vecteur colonne $m \times 1$, Y est une matrice $m \times n$, le logiciel trace alors chaque colonne de Y en fonction de celle de X, soit les points $(X(i), Y(i, j))$, pour $i = 1, 2, \cdots, m$ pour chaque j,
- X et Y sont des matrices de même dimension $m \times n$, le logiciel trace chaque j-ième collone de Y versus la j-ième colonne de X.

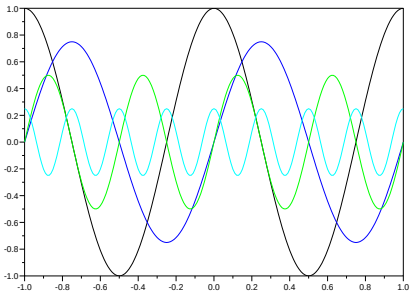
La fonction `subplot(m,n,k)` divise la fenêtre graphique en une matrice de sous-graphiques m lignes et n colonnes. L'indice k permet de numéroter chaque sous-graphique de 1 à mn, ligne après ligne de la gauche vers la droite.

```
%Graphique sous Matlab
clf;
xx = (-1):0.01:1;
yy = cos(2*pi*xx);
zz = 0.75*sin(2*pi*xx);
ww = 0.5*sin(4*pi*xx);
vv = 0.25*cos(8*pi*xx);
plot(xx',[yy',zz',ww',vv']);
```



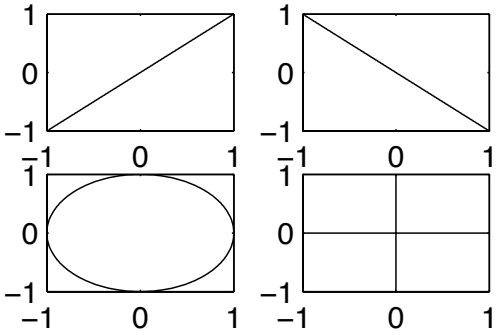
Graphique sous Matlab

```
//Graphique sous Scilab
clf;
xx = (-1):0.01:1;
yy = cos(2*%pi*xx);
zz = 0.75*sin(2*%pi*xx);
ww = 0.5*sin(4*%pi*xx);
vv = 0.25*cos(8*%pi*xx);
plot2d(xx',[yy',zz',ww',vv']);
```



Graphique sous Scilab

```
%utilisation de subplot
clf;
xx = (-1):0.01:1;
nn = size(xx,2);
subplot(2,2,1);
plot(xx,xx);
subplot(2,2,2);
plot(xx,-xx);
subplot(2,2,3);
plot(cos(pi*xx),sin(pi*xx));
subplot(2,2,4);
plot([[-1;1],[0;0]],...
     [[0;0],[-1;1]]);
```



Utilisation de `subplot`

A.7.1 Scilab

Pour connaître toutes les fonctionalités graphique de Scilab, le plus simple est de l'obtenir en ligne per :

```
help Graphics
```

On peut néanmoins se contenter des informations suivantes dans un premier temps.

Par défaut, l'utilisation successive de `plot2d` superpose les graphiques. Pour effacer un graphique on utilise `clf`. La fonction `plot2d` possède 4 variantes :

Scilab	
`plot2d1`	voir option de `plot2d`
`plot2d2`	graphe en marche d'escalier
`plot2d3`	graphe en barres verticales
`plot2d4`	graphe en lignes de courant

De plus, les fonctionalités de `plot2d`, ainsi que des versions `plot2dx`, peuvent être modifiées par un ensembles d'options

plot2d(X,Y,options)

Il y a deux manières de présenter ces options.

Présentation ancienne :	
`options = style,"xyz",leg,rect,nax`	
`style`	`[i,j,...]`
`"xyz"`	`x=0,1 y=0 :8 z=1 :5`
`leg`	`"leg1@leg2@..."`
`rect`	`[xmin,ymin,xmax,ymax]`
`nax`	`[nx,Nx,ny,Ny]`

Il y a autant d'indices dans `style` qu'il y a de courbes à dessiner. La signification des indices `i,j...` est donnée plus loin. Si `x=0`, aucune légende n'est dessinée et `leg=""`, si `x=1`, il y a autant de légendes que de courbes à dessiner. La signification de `"yz"` est donnée plus loin. Le vecteur `rect` délimite les bornes minimales et maximales, à la fois des axes horizontaux et verticaux. Le vecteur `nax` donne le nombre de graduations principales `Nx,Ny` et le nombre de sous-graduations `nx,ny` entre deux graduations principales.

Présentation moderne :	
`options = clef1=valeur1,clef2=valeur2,...`	
clef	**valeur**
`style`	`[i,j,...]`
`logflag`	`"nn" "nl" "ln" "ll"`
`frameflag`	`f = 0 :8`
`axesflag`	`a = 0 :5`
`leg`	`"leg1@leg2@..."`
`rect`	`[xmin,ymin,xmax,ymax]`
`nax`	`[nx,Nx,ny,Ny]`

– `style` : un vecteur `[i,j,...]` de nombres entiers de même dimension que le nombre de courbes à dessiner ; l'indice `i` concerne par exemple la première courbe. Si l'indice est positif, la courbe est dessinée en trait plein de la couleur correspondant au tableau :

Couleur des courbes :			
1	black	6	magenta
2	blue	7	yellow
3	green	8	white
4	cyan	...	...
5	red	32	gold

(obtenu par `getcolor()`)

Si l'indice est négatif, la courbe est dessinée point par point en utilisant le symbol correspondant :

Type du marqueur :			
0	•	-5	♢
-1	+	-6	⊗
-2	×	-7	▽
-3	∗	-8	♣
-4		-9	○

(obtenu par `getmark()`)

On verra ultérieurement des fonctions graphiques de plus bas niveau, mais on retiendra déjà les fonctionalité suivantes de `xset`.

Fonction : `xset(nom,valeur)`	
nom	**valeur**
`"default"`	pas de valeur
`"line style"`	`k = 1 :6`
`"mark size"`	`l = 0 :5`
`"thickness"`	`m = 1,2,3,...`

`"mark size"` détermine la taille du marqueur (l=0 pour la plus petite taille). `"thickness"` détermine l'épaisseur du trait (m=1 pixel, m=2 pixels,...). `"line style"` détermine la forme du trait suivant le tableau :

Forme du trait :	
`k=1`	———
`k=2`	– – – – –
`k=3`	· · · · · · · · · ·
`k=4`	– · – · – · –
`k=5`	– ·· – ·· – ··
`k=6`	

(obtenu par getlinestyle())

- `logflag` : Echelle linéaire `n` ou logarithmique `l`.
- `frameflag` : Cette option permet de calculer les bornes du dessin. Elles peuvent être calculées automatiquement si l'option `rect` n'est pas utilisée. Les différentes possibilités sont données dans le tableau :

Bornes du dessin :	
`f=0`	calculées précédemment
`f=1`	données par `rect`
`f=2`	calculées par min/max
`f=3`	isométriques sur `rect`
`f=4`	isométriques sur min/max
`f=5`	idem à 1 plus harmonieux
`f=6`	idem à 2 plus harmonieux
`f=7`	idem à 1, tout est redessiné
`f=8`	idem à 2, tout est redessiné

Les bornes du dessin peuvent être calculées, ou bien par `rect`, ou bien par les bornes des données. Dans l'option isométrique, les axes horizontaux et verticaux ont la même echelle. Pour les deux dernières options, tous les graphiques précédents sont redessinés.

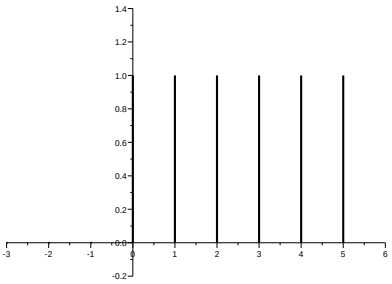
- `axesflag` : Cette option permet de placer les axes et eventuellement un cadre autour du graphique. Les différentes options sont données dans le tableau :

Placement de axes :	
`a=0`	pas d'axes, pas de cadre
`a=1`	axes bas/gauche, un cadre
`a=2`	pas d'axes, un cadre
`a=3`	axes bas/droit, un cadre
`a=4`	axes centrés, pas de cadre
`a=5`	axes en `x=y=0`, pas de cadre

- `leg` : Cette option permet d'afficher une légende `"leg1"`, `"leg2"`, ...à chaque courbe.
- `rect` : Un vecteur ligne à 4 entrées donnant les bornes du dessin : horizontalement par `[xmin,xmax]`, verticalement par `[ymin,ymax]`.
- `nax` : Cette option permet de controler le nombre de graduations principales `Nx,Ny` et secondaires `nx,ny` affichées sur les axes. Ne fonctionne qu'avec l'option `axesflag=1`.

Le graphe suivant représente la fonction de Heaviside discrète sur l'intervalle de temps

```
xx = [-3,-2,-1,0,1,2,3,4,5]
yy = [ 0, 0, 0,1,1,1,1,1,1]
```



Fonction de Heaviside discrète

dans le code, on commence par tracer le graphe de `yy` en traits épais `thickness=5` puis les axes en traits minces `thickness=1`.

```
//Fonction de Heaviside discrete
clf;clear;
xx = (-3):1:5; yy = bool2s(xx>=0);
xset("thickness",5);
```

```
plot2d3(xx',yy',[1],...
        style=1,leg="",...
        frameflag=1,axesflag=0,...
        rect=[-3,-0.2,6,1.4]);
xset("thickness",1);
plot2d(0,0,[1],style=1,leg="",...
       frameflag=1,axesflag=5,...
       rect=[-3,-0.2,6,1.4]);
```

Dans le graphe suivant, on utilise simultanément `plot2d2` et `plot2d3` pour obtenir un diagramme en barres :

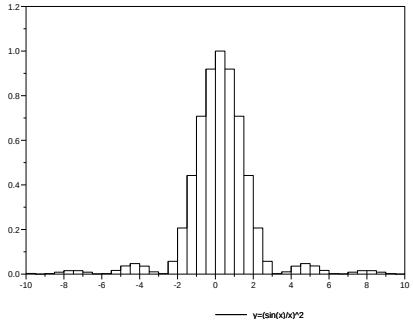


Diagramme en barres

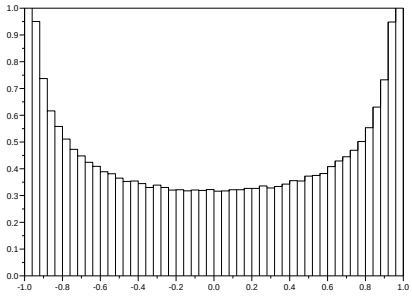
```
//Diagramme en barres
clf; clear;
xx = (-10):0.5:10;
dd = bool2s(xx==0);
yy = ((sin(xx)+dd)./(xx+dd))^2;
style = 1;
opt = "111";
leg = "y=(sin(x)/x)^2";
rect = [-10,0,10,1.2];
plot2d2(xx',yy',style,...
        opt,leg,rect);
plot2d3(xx',yy',style,...
        opt,leg,rect);
```

Scilab possède beaucoup d'autres fonctions graphiques plus spécialisées :

- Histogramme : visualise la fréquence d'apparition d'une suite de valeurs dans un ensemble de classes.
- Courbes de viveau : trace des lignes de hauteur constante sur un graphe 3D paramétrique $z = f(x, y)$.
- Surfaces par niveaux de couleurs : remplace les courbes de niveau par un dégradé de couleurs.
- Champ de vecteurs : trace des lignes discrètes de courant d'un champ de vecteur du plan.

Histogramme :	
`histplot(N,data,options)`	
`N`	nombre de classes
`N`	ou vecteur de classes
`data`	données à classer
`options`	options de `plot2d`

Le graphe suivant représente un histogramme d'une orbite "typique" sous l'action du système dynamique $x_{n+1} = 1 - 2x_n^2$ partant d'une donnée aléatoire $x_0 \in [-1, 1]$.

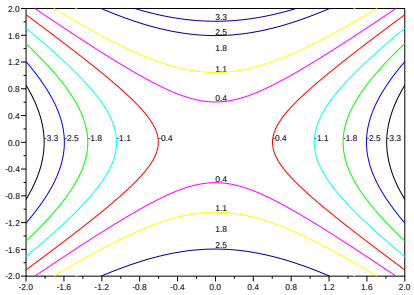


Histogramme dynamique

```
//Histogramme dynamique
clear; stacksize(2000000);
nn = 400000; yy = zeros(nn,1);
xx = 2*rand()-1;
for (ii = 1:nn),
  yy(ii)=xx; xx = 1-2*xx^2;
end
nb = 50; //nb de classes
style = 1; opt = "011";
leg = ""; rect = [-1,0,1,1];
clf; histplot...
  (nb,yy,style,opt,leg,rect);
```

Courbes de niveau	
`contour2d(x,y,z,N,options)`	
`x`	vect. m des abscisses
`y`	vect. n des ordonnées
`z`	mat. $m \times n$ des hauteurs
`N`	nb de courbes de niveau
`N`	vect. des courbes de niveau
`options`	options de `plot2d`

Le graphe suivant représente 10 courbes de niveau sur la surface $z = x^2 - y^2$ définie sur $x \in [-2, 2]$ et $y \in [-2, 2]$.



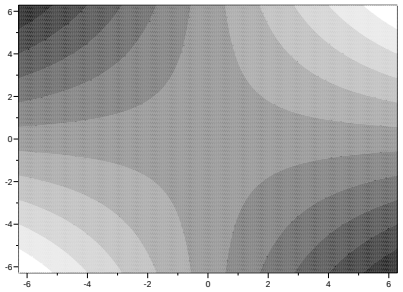
Courbes de niveau

```
//Courbes de niveau
xx = (-2):0.01:(2);
yy = (-2):0.01:(2);
cc = size(xx,2);
ll = size(yy,2);
zz = ones(ll,1)*(xx^2)...
     -(yy^2)'*ones(1,cc);
nz = 10;
style = 1:10; opt = "011";
leg = ""; rect = [-2,-2,2,2];
clf; xset("fpf","%.1f");
contour2d(xx,yy,zz,nz,...
          style,opt,leg,rect);
```

La fonction `xset("fpf",format)` a été utilisée pour formater les hauteurs des courbes de niveau ; `format = "%.1f"` par exemple fait référence au formatage des nombres dans le langage C.

Surface par niveaux de couleurs :	
`Sgrayplot(x,y,z,options)`	
`x`	vect. m des abscisses
`y`	vect. n des ordonnés
`z`	mat. $m \times n$ des hauteurs
`options`	`["xyz",leg,rect,nax]`

Le graphe suivant visualise la surface $z = \cos(x)\sin(y)$ sur le domaine $x \in [-2\pi, 2\pi]$, $y \in [-2\pi, 2\pi]$ par niveaux de gris au lieu de couleurs.



Surface par niveaux de gris

```
clf;clear;
stacksize(2000000);
x = 2*%pi*((-1):0.01:1);
y = 2*%pi*((-1):0.01:1);
z = y'*x;
//modifie la table des couleurs
rr = (0:10)'/10; gg = rr; bb = rr;
xset("colormap",[rr,gg,bb]);
rect = [-2*%pi,-2*%pi,2*%pi,2*%pi];
Sgrayplot(x,y,z,"011",rect);
```

Champ de vecteur :	
`champ(x,y,u,v,options)`	
`x`	vect. m des abscisses
`y`	vect. n des ordonnés
`u`	mat. $m \times n$ des comp. sur x du champ de vecteur
`v`	mat. $m \times n$ des comp. sur y du champ de vecteur
`options`	`[arfact,rect,"xyz"]`

A.8 Polynômes

Le traitement des polynômes est différent en Matlab et en Scilab. Un polynôme $a_0 + a_1 x + \cdots + a_n x^n$ en Matlab est représenté simplement par le vecteur ligne `[a_n,...,a_1,a_0]`. Scilab propose une nouvelle structure et permet de la généraliser aux fractions rationnelles.

Scilab	
`C,P`	vecteur lig.
`R`	vecteur col.
`P`	polynôme
`P = poly(C,"x","coeff")`	
`P = poly(R,"x","roots")`	

Dans l'expression

```
P = poly(C,"x","coeff")
```

`P` est un polynôme d'indéterminée `"x"` dont les coefficients sont donnés par

```
C = [a_0,a_1,...,a_n]
```

(remarquer l'ordre inverse par rapport à Matlab). L'indéterminée `"x"` peut être remplacée par une autre lettre ; les polynômes obtenus sont alors incompatibles. Dans l'expression

```
P = poly(R,"x","roots")
```

`P` est un polynôme unitaire dont les racines sont données par les composantes du vecteur `R`. Le code suivant, par exemple, représente le polynôme $(1+x)^2$:

```
%Matlab
P = [1,2,1];
//Scilab
P = poly([1,2,1],"x","coeff");
```

Matlab peut aussi construire des polynômes à partir de ces racines. On utilise :

Matlab	
`R`	vect. col. des racines
`P`	vecteur ligne
`P = poly(R)`	$(x-r_1)\cdots(x-r_n)$

Par exemple, le polynôme $P = -z^2 + z^3$ est représenté par le code

```
%Matlab
P = poly([0;0;1]);
//Scilab
P = poly([0,0,1],"z","roots");
```

Scilab possède enfin une autre possibilité de construction de polynômes en utilisant un peu d'algèbre. Par exemple $(1+x)^2$ s'écrit sous la forme :

```
//Scilab
x = poly(0,"x","roots");
P = 1+2*x+x^2;
```

Les deux logiciels possèdent néanmoins beaucoup de fonctions semblables :

Matlab	
`R=roots(P)`	vect. des rac.
`P=poly(A)`	$det(xI - A)$
`s=polyval(P,x)`	éval. P en x
`C=conv(A,B)`	prod. de pol.
`[Q,R]=deconv(A,B)`	div. euclid.
`[R,P,K]=residue(A,B)`	frac. ration.
`[B,A]=residue(R,P,Q)`	frac. ration.

Scilab	
`P,Q,U,V`	polynôme
`A`	matrice carrée
`R`	vecteur colonne
`R=roots(P)`	racines de P
`P=poly(A,"x")`	$P = det(xI - A)$
`P*Q`	prod. de pol.
`[U,V]=pdiv(P,Q)`	div. eucl.

```
//Scilab
//Racines d'un polynome
P = poly([6,-5,1],"s","coeff")
R = roots(P) //R = [2;3]
Q = poly(R,"z","roots")

//produit de polynomes
Z = poly(0,"z","roots");
P =1+Z; Q = 1-Z;
P*Q // 1-Z^2

//division euclidienne
s = poly([0,1],"s","coeff");
A = s^2+3*s+1; B=s-1;
[R,Q] = pdiv(A,B) // R=5 Q=4+s
R+B*Q // ans=1+3s+s^2
```

A.9 Programmation

Matlab et Scilab, en dehors d'être des super-calculateur, disposent aussi d'un langage de programmation simple mais suffisament complet. Ils sont sur ce point très semblables. Contrairement aux langages compilés (C, Fortran, ...), les variables ne sont pas déclarées : le type (entier, complexe, matrice, liste,...), la taille ne sont pas précisés la première fois que le compilateur rencontre l'objet. C'est un inconvénient en génie logiciel car les erreurs sont difficilement repérables ; c'est un atout en expérimentation et prototypage.

A.9.1 Les instructions de contrôle

On distingue les boucles et les instructions conditionnelles.

Les boucles :

Matlab Scilab	

`for var =`
`expr,`
`  instr ;`
`  instr ;`
`  ...`
`end`
	`while expr,`
	`  instr ;`
	`  instr ;`
	`  ...`
	`end`

Dans le cas de l'instruction `for`, la variable `var` prend successivement les valeurs se trouvant dans chaque colonne du vecteur ou matrice `expr` et à chaque prise de valeurs, les instructions `instr` sont exécutés les unes après les autres.

```
clear;
mot = ['n','o','e','l']; tom = [''];
for ii = size(mot,2):(-1):1,
    tom = [tom,mot(ii)];
end
tom // ans = ['l','e','o','n']
```

Pour l'instruction `while`, Les instructions `instr` sont exécutées tant que l'expression booléenne `expr` reste vraie. En Matlab, le type "vrai" est représenté par un réel strictement positif (1 par exemple).

```
clear;
N=7; dd = 2; resultat = zeros(N,1);
fibo = ones(1,dd); ii = 1;
while ii <= N,
    resultats(ii) = fibo(dd);
    fibo = [fibo(2:dd),sum(fibo)];
    ii = ii + 1;
end
resultats
//ans [1,2,3,5,8,13,21]
```

Les instructions conditionnelles :

Matlab Scilab

`if expr1,`
`  instr1 ;`
`elseif expr2,`
`  instr2 ;`
else
`  instr3 ;`
`end`

Dans les instructions conditionnelles `if else end` ou `if elseif else end`, `else` est facultatif et `elseif` peut être répété plusieurs fois. Si l'expression `expr1` n'est pas réalisée (`%F` en Scilab, 0 en Matlab), l'une des deux instructions `instr2` ou `instr3` est exécutée selon la valeur de `expr2`, sinon `instr1` est exécutée.

Matlab	Scilab

`select expr,`
`  case expr1,`
`  instr1 ;`
`  case expr2,`
`  instr2 ;`
`  else,`
`  instr3 ;`
`end`
	`switch expr,`
	`  case expr1,`
	`  instr1 ;`
	`  case expr2,`
	`  instr2 ;`
	`  else,`
	`  instr3 ;`
	`end`

A.9.2 Les fonctions

La syntaxe de construction et d'utilisation des fonctions est identique en Matlab et Scilab. Il y a cependant un point important, concernant le chargement en mémoire vive, qui diffère sensiblement. Pour Matlab, chaque fonction `maFonction` doit être sauvegardée dans un unique fichier qui s'appellera forcément `maFonction.m` ; l'appelle à cette fonction se fait simplement par

```
var_sortie = maFonction(var_entrée) ;
```

Pour Scilab, un fichier `maBibli.sci`, (le nom et l'extension `.sci` sont sans importance), peut contenir plusieurs fonctions `maFonction1`, `maFonction2` ; son chargement en mémoire se fait par l'instruction

```
getf('mabBibli.sci') ;
```

et les fonctions qui s'y trouvent sont appellées comme précédemment.

Syntaxe : La première ligne doit respecter le format suivant : le mot clef `function`, un vecteur ligne d'arguments en sortie `Y=[y1,y2,...]`, le signe `=`, le nom de la fonction `maFonction`, une liste d'arguments en entrée `X=x1,x2,...`, un retour à la ligne, une suite d'instructions `instr`, pour Scilab seulement, le mot clef `endfunction`.

Matlab Scilab
`Y=[y1,y2,...]` `X=x1,x2,...`

`function Y = maFonction(X)`
`  instr1 ;`
`  instr2 ;`
`endfunction //Scilab seul`

Sous Matlab, les lignes "commentaires" qui suivent directement la déclaration `function Y = mafFonction(X)` sont visibles lorsqu'on exécute la commande `help maFonction`.

```
//Scilab
function [r1,r2] = equa2D(P)
//P      : polynome du second degre
//r1, r2 : racines

if typeof(P) ~= "polynomial",
   error("l"+code2str(53)+...
   "argument en entree"+...
   "doit etre un polynome")
end
if degree(P) ~= 2,
   error("le polynome"+...
   "n"+code2str(53)+"est pas"+...
   "du second degre")
end
pp = coeff(P);
a=pp(3); b = pp(2); c = pp(1);

delta = b^2-4*a*c;
if delta > 0,
   r1 = (-b+sqrt(delta))/(2*a);
   r2 = (-b-sqrt(delta))/(2*a);
else
   r1 = (-b+%i*sqrt(-delta))/(2*a);
   r2 = (-b-%i*sqrt(-delta))/(2*a);
end
//[u,v] = equa2D((%s-1)*(%s-2))
//u = 1; v = 2;
//[u,v] = equa2D(%s^2+1)
```