

ANNEXE 1. PRISE EN MAIN D’UN LOGICIEL DE CALCUL SCIENTIFIQUE (MATLAB, SCILAB)

MATLAB et Scilab sont deux logiciels de calcul scientifique très similaires. MATLAB est géré par la société MATHWORKS ; il est plus professionnel et payant. Scilab est géré par le Consortium Scilab (INRIA, ENPC) ; il est du domaine public 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 90 (voir le chapitre 5). En général, un langage interprété exécute les instructions beaucoup plus lentement qu’un langage compilé. Un langage interprété se justifie, en revanche, par un temps de conception et de débogage extrêmement réduit. Par ailleurs, dans les deux cas, une très large collection de fonctions et de documentation est disponible.

1.1. Démarrer

Il est tout d’abord indispensable de bien savoir naviguer dans son environnement de programmation : savoir se repérer dans la structure des fichiers, changer de répertoire, créer des *scripts*, les exécuter, connaître les variables d’environnement, etc.

MATLAB	Scilab	Remarques
<code>pwd</code>	<code>pwd</code>	affiche le répertoire courant
<code>cd</code>	<code>cd</code>	change de répertoire
<code>ls</code>	<code>ls</code>	affiche la liste des fichiers et répertoires
<code>edit('script.m')</code>	<code>scipad('script.sce')</code>	ouvre ou crée un fichier
<code>doc sin</code>	<code>help sin</code>	affiche l’aide en ligne de <code>sin</code>
<code>whos</code>	<code>whos()</code>	liste des variables courantes
<code>which conv</code>		chemin d’accès à la fonction <code>conv</code>

```
// Scilab
pwd // 'c:\Documents and Settings'
cd 'c:\Program Files'
ls('SCI/macros/util/*.sci')
scipad('fichier_script.sce')
```

```
% MATLAB
pwd % 'c:\Documents and Settings'
cd 'c:\Program Files'
ls('MATLABROOT\toolbox\matlab\*')
edit('fichier_script.m')
```

On constate que le début d’un commentaire est signalé par % sur MATLAB, par // sur Scilab et s’arrête en fin de ligne. Une ligne ne peut comporter qu’une seule instruction en principe. On peut séparer plusieurs instructions sur une même ligne par une virgule « , » ou un point-virgule « ; ». Il faut noter cependant que « ; » a une autre signification : le résultat d’une instruction non terminée par « ; » est affiché en retour ; cela peut être gênant lorsque l’on exécute un *script*, mais utile lorsque l’on se sert de ces logiciels comme simple calculateur. On retiendra enfin qu’une instruction peut être coupée par un saut de ligne, quitte à terminer la ligne d’instruction par 3 points de suspension.

```
// Scilab
1+2+3+4+5+ ...
6+7+8+9 // résultat 45
```

```
% MATLAB
1+2+3+4+5+ ...
6+7+8+9 % résultat 45
```

Si l’on désire exécuter une suite d’instructions, il est préférable d’écrire d’abord un fichier, appelé « *script* », avec un éditeur quelconque, puis de l’exécuter globalement par la suite.

```
// Scilab
scipad('fichier.sce') // crée fichier.sce
exec('fichier.sce') //exécute fichier.sce
```

```
% MATLAB
edit fichier.m % crée fichier.m
fichier % exécute fichier.m
```

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 la commande `exec nom_fichier.sce` sur Scilab et seulement le nom du fichier `nom_fichier`, sans extension, sur MATLAB. 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. Sur MATLAB, il n’existe pas de fonction équivalente ; il faut manuellement augmenter la taille de la mémoire virtuelle.

1.2. Un calculateur puissant

MATLAB et Scilab peuvent être utilisés en première approche comme de simples calculettes. Nous verrons par la suite qu’ils possèdent un type de données très performant : le type « vecteur » ou le type « matrice ».

Opérateurs arithmétiques et format

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

MATLAB, Scilab	
<code>+</code>	addition
<code>-</code>	soustraction
<code>*</code>	multiplication
<code>/</code>	division
<code>^</code>	puissance
<code>()</code>	parenthèses

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

Constantes prédéfinies

Un certain nombre de constantes prédéfinies peuvent être utilisées dans une instruction.

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

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

On constate que Scilab possède des constantes booléennes `%F`, « faux », et `%T`, « vrai », qui se comportent comme des scalaires 0 et 1 dans des calculs arithmétiques `%T+1=2`. MATLAB utilise les scalaires 0 et 1 avec les mêmes conventions.

Types des données

Les types de données les plus importants sont les types scalaires (entiers, réels ou complexes), le type booléen et le type caractère. Pour connaître le type de la donnée, on utilisera

MATLAB	Scilab	Remarques
<code>class(var)</code>	<code>typeof(var)</code>	type de la variable <code>var</code>

```
// Scilab
var=(2>3);typeof(var) //ans = boolean
typeof('a')           //ans = string
```

```
% MATLAB
var=(2>3); class(var) % ans = logical
class('a')           % ans = char
```

Fonctions standard

MATLAB et Scilab possèdent des bibliothèques très riches de fonctions aussi bien usuelles, comme les fonctions trigonométriques `cos`, `sin`, etc., que spécialisées en traitement du signal, statistique, etc. La liste suivante n'est pas exhaustive.

MATLAB / Scilab	
<code>acos</code>	fonction inverse à <code>cos</code>
<code>asin</code>	fonction inverse à <code>sin</code>
<code>atan</code>	fonction inverse à <code>tan</code>
<code>cos</code>	fonction cosinus
<code>exp</code>	fonction exponentielle
<code>log</code>	fonction logarithme
<code>sin</code>	fonction sinus
<code>sqrt</code>	fonction racine carrée
<code>tan</code>	fonction tangente

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

Ces dernières fonctions retournent des valeurs en double précision. Les suivantes retournent des valeurs entières.

MATLAB / Scilab	
<code>ceil</code>	partie entière supérieure
<code>floor</code>	partie entière inférieure
<code>round</code>	plus proche entier
<code>sign</code>	fonction signe (-1,0,1)
<code>fix</code>	<code>sign(x).floor(abs(x))</code>

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

Les fonctions suivantes sont, en revanche, différentes suivant que l'on utilise MATLAB ou Scilab.

MATLAB	Scilab	Remarques
<code>abs</code>	<code>abs</code>	module
<code>angle</code>	<code>phasemag</code>	argument
<code>conj</code>	<code>conj</code>	conjugué
<code>imag</code>	<code>imag</code>	partie imag.
<code>real</code>	<code>real</code>	partie réelle
<code>mod</code>	<code>pmodulo</code>	reste eucl. <code>floor</code>
<code>rem</code>	<code>modulo</code>	reste eucl. <code>fix</code>

```
// Scilab
ii = exp(%i*%pi/2) // ii = 6.123D-17 + i
abs(ii) // ans = 1.
[phase,gain] = phasemag(ii)
// gain = 0. phase = 90.
imag(log(2*exp(%i*%pi/2)))
// ans = 1.5707963
atan(imag(ii),real(ii))
// ans = 1.5707963
% MATLAB
ii = exp(i*pi/2) % ii = 0.000 + 1.0000i
angle(ii) % ans = 1.5708
```

1.3. Variables

On vient d'utiliser une variable `ii` dans le *script* précédent. Il faut noter que les règles de construction d'un nom de variable diffèrent entre MATLAB et Scilab.

Un nom de variable en MATLAB est constitué de chiffres, de lettres et de caractères spéciaux : `1 2 ... 9 a b c ... x y z _`. Un nom doit commencer par une lettre. Seuls les `namelengthmax = 63` premiers caractères sont pris en compte. Majuscules et minuscules sont prises en compte. Un nom de variable en Scilab est constitué de chiffres, de lettres et de caractères spéciaux :

1 2 ... 9 a b c ... x y z % _ # ! \$?. Un nom ne peut pas commencer par un chiffre. Le caractère % ne peut apparaître qu’en première position. Seuls les 24 premiers caractères sont pris en compte. Majuscules et minuscules sont prises en compte.

<pre>// Scilab %eps, un_nom, #123 // noms valides 1abc, A%A, .C // noms invalides</pre>	<pre>% MATLAB AaA, A_a_A, A1a1A % noms valides _abc, 123 % noms invalides</pre>
--	---

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* que l’on exécute, sont globales et sont donc conservées en mémoire.

1.4. Opérateurs logiques

MATLAB et Scilab offrent les opérateurs logiques usuels de tout langage de programmation.

<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th colspan="2">MATLAB / Scilab</th> </tr> </thead> <tbody> <tr><td>==</td><td>égal à</td></tr> <tr><td>~=</td><td>différent de</td></tr> <tr><td><</td><td>inférieur à</td></tr> <tr><td>></td><td>supérieur à</td></tr> <tr><td><=</td><td>inférieur ou égal à</td></tr> <tr><td>>=</td><td>supérieur ou égal à</td></tr> <tr><td>~</td><td>non</td></tr> <tr><td>&</td><td>et</td></tr> <tr><td> </td><td>ou</td></tr> </tbody> </table>	MATLAB / Scilab		==	égal à	~=	différent de	<	inférieur à	>	supérieur à	<=	inférieur ou égal à	>=	supérieur ou égal à	~	non	&	et		ou	<pre>// Scilab bool = (1.5<2)&(1.5>=1) // bool = T 2 == 3 // ans = F ~[0,1,0,0] % ans = [T,F,T,T] % MATLAB bool = (1.5<2)&(1.5>=1) % bool = 1 2 == 3 % ans = 0 ~[0,1,0,0] % ans = [1,0,1,1]</pre>
MATLAB / Scilab																					
==	égal à																				
~=	différent de																				
<	inférieur à																				
>	supérieur à																				
<=	inférieur ou égal à																				
>=	supérieur ou égal à																				
~	non																				
&	et																				
	ou																				

1.5. Vecteurs et matrices

Une des grandes forces de MATLAB et Scilab est de pouvoir travailler très rapidement sur des vecteurs, ou plus généralement sur des matrices, de deux ou plusieurs dimensions. Par exemple, il est possible d’appliquer une même opération simultanément sur toutes les composantes d’un vecteur, `[1,2,3,4,5].^2 // ans = 1 4 9 16 25`, sans pour autant écrire une boucle de calcul.

Construction

En MATLAB ou Scilab, 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 comme par exemple

<pre>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]</pre>	$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}, \quad B = [1 \ 2 \ 3]$ $C = {}^t [1, 2, 3], \quad D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}.$
---	---

On peut aussi construire des matrices par blocs par concaténation comme par exemple

<pre>A = [1,2;3,4], B = [5;6] C = [7,8], D = 9 E = [A,B;C,D]</pre>	$E = \begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \begin{bmatrix} 5 \\ 6 \end{bmatrix} \\ \begin{bmatrix} 7 & 8 \end{bmatrix} & \begin{bmatrix} 9 \end{bmatrix} \end{bmatrix}.$
--	--

Notation « deux points »

Il existe une syntaxe très pratique pour construire un vecteur-ligne : l’opérateur 2 points « : ». Lors que le pas n’est pas indiqué, il est sous-entendu égal à 1.

MATLAB / Scilab
debut : pas : fin
debut : fin

```
x = 1:10 // x = [1 2 3 4 5 6 7 8 9 10]
x = 1:1:10 // x = [1 2 3 4 5 6 7 8 9 10]
x = 1:2:10 // x = [1 3 5 7 9]
x = 1:-2:-10 % x = [1 -1 -3 -5 -7 -9]
x = 0:0.05:0.13 % x = [0 0.05 0.1]
```

Extraction

Un vecteur (-colonne ou -ligne) est en fait une matrice d’un type particulier ($m \times 1$ ou bien $1 \times n$). On accède aux données d’un vecteur à partir de l’indice 1 : $V(1)$ désigne la première coordonnée de V , $V(2)$, la deuxième coordonnée, etc. Pour une matrice A de dimension $m \times n$, $A(i, j)$ désigne l’entrée de A à la i -ième ligne et à la j -ième colonne. Par exemple, si $A = [0 : 2 : 16]$, alors $A(1)=0$, $A(4)=6$, $A(9)=16$, si $B = [1, 2, 3 ; 4, 5, 6]$, alors $B(1,1)=1$ et $B(2,3)=6$. On peut aussi extraire des sous-matrices $A(i, j)$ en prenant cette fois-ci pour i et j des vecteurs d’indices (ligne ou colonne). Si i ou j est égal au caractère deux points « : », il est sous-entendu que toutes les lignes ou toutes les colonnes sont retenues. Pour une matrice A , $A(:)$ désigne le vecteur-colonne obtenu en mettant bout à bout toutes les colonnes de A .

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

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

```
C = A(:, [2;4])
D = A(1, :)
```

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

Modifier le contenu d’une matrice se fait de la même manière.

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

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

Le tableau suivant récapitule les différentes constructions que l’on peut réaliser sur un vecteur ou une matrice.

MATLAB / Scilab		MATLAB / Scilab	
V	vecteur de dimension m	A	matrice de dimension $m \times n$
ii	indice : $1 \leq ii \leq m$	ii, jj	$ii \leq m, jj \leq n$
V(ii)	ii -ième composante de V	A(ii, jj)	entrée (ii, jj) de A
II	$II = [i_1, \dots, i_r]$	II	$II = [i_1, \dots, i_p]$
V(II)	$[V(i_1), V(i_2), \dots, V(i_r)]$	JJ	$JJ = [j_1, \dots, j_q]$
		A(II, JJ)	matrice de dimension $p \times q$
		A(: , JJ)	matrice de dimension $m \times q$
		A(II, :)	matrice de dimension $p \times n$
		A(:)	vecteur-colonne de dim. mn

Les vecteurs-ligne **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 comme par exemple

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

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

On peut aussi extraire des données d’une matrice en utilisant comme indices 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 entrées 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. Les colonnes sont parcourues les unes après les autres, de haut en bas.

MATLAB / Scilab	
V	vecteur de dimension m
W	vecteur booléen de dim m
V(W)	V(i) si W(i) = vrai

MATLAB / Scilab	
A	matrice m × n
B	matrice booléenne m × n
A(B)	A(i,j) si B(i,j) = vrai

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

$$V = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10]$$

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

```
// Scilab
A = [1+%i,1,1-%i;1,0,-1;-%i,%i,0]
B = A(%T,%F,%T;%F,%T,%F;%T,%F,%T)
% 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)
```

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

$$A(B) = {}^t [1+i, -i, 0, 1-i, 0]$$

On retiendra les fonctions `logical`, spécifique à MATLAB et `bool2s`, spécifique à Scilab.

MATLAB	
A	matrice scalaire
B = logical(A)	matrice booléenne

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

Pour MATLAB, $B(i,j) = 1$ si et seulement si $A(i,i) \neq 0$, $B(i,j) = 0$ sinon. Pour Scilab, $C(i,j) = 0$ si $A(i,j) = \%F$ et $C(i,j) = 1$ si $A(i,j) = \%T$.

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 × n
size(A)	le vecteur [m,n]
size(A,1)	le réel m
size(A,2)	le réel n

```
A = [1:2:9;9:-2:1]
V = A(2,:)
dim_A = size(A) // dim_A=[2,5]
dim_V = length(V) // dim_V=5
nb_lignes = size(A,1) // 2
nb_colonnes = size(A,2) // 5
```

Opérations matricielles

Le tableau ci-dessous résume toutes les opérations matricielles avancées que ces deux logiciels acceptent. Il faut noter l'importance des opérateurs « .* », « .^ », « ./ » qui permettent d'agir simultanément, composante par composante. Cette technique de calculs parallèles permet d'éviter d'écrire des boucles qu'il faut proscrire absolument.

MATLAB / Scilab					
A,B	matrices $m \times n$	s*A	produit par un scalaire	A.*B	produit terme à terme
C	matrice $n \times p$	A/s	division par un scalaire	A./B	division terme à terme
s	scalaire	A+B	addition matricielle	s./A	division terme à terme
A'	transposé, conjugué	A*C	produit matriciel	A.^B	puissance terme à terme
A.'	transposé	A^s	puissance d'une matrice	s.^A	puissance terme à terme
				A.^s	puissance terme à terme

Par exemple, le produit scalaire euclidien de deux vecteurs-ligne U et V s'écrit très simplement par $U*V'$; ici V' représente le vecteur-colonne transposé de V . Par exemple, si $U = [1,2,3,4]$, $V = [1,-1,3,-2]$, alors le produit scalaire est aussi bien donné par $U*V'$ que par $\text{sum}(U.*V)$ et vaut 0 dans les deux cas. Dans l'exemple suivant, bien remarquer la différence d'utilisation des constructions $V.^2$ et A^2 .

```
V = 1:10 // V=[1,2,3,4,5,6,7,8,9,10]
W = V.^2
// W=[1,4,9,16,25,36,49,64,81,100]
A = [0,0,0,1;1,0,0,0;0,1,0,0;0,0,1,0]
B = A^4
```

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

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), où chaque entrée (i,j) est le résultat booléen $A(i,j) < B(i,j)$; si s est un scalaire, $s < A$ est aussi une matrice booléenne d'entrée (i,j) égale au résultat booléen $s < A(i,j)$. Les opérations booléennes sur les matrices sont résumées dans le tableau suivant.

MATLAB / Scilab			
A,B	matrice $m \times n$	A==B	$A(i,j) == B(i,j)$
s	scalaire	A~=B	$A(i,j) \neq B(i,j)$
~ A	$\sim A(i,j)$	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<=B	$A(i,j) \leq B(i,j)$
		A>=B	$A(i,j) \geq B(i,j)$

Par exemple, le code $V = 1 : 10$, $W = (V >= 6)$ donne $W = [0,0,0,0,0,1,1,1,1,1]$ sur MATLAB et $W = [F,F,F,F,F,T,T,T,T]$ sur Scilab. Les fonctions suivantes permettent de savoir si un vecteur ou une 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	
V	vecteur-ligne
A	matrice $m \times n$
any(V)	$\sup_i \text{sign} V(i) $
any(A,1)	$\sup_i \text{sign} A(i,j) $
any(A,2)	$\sup_j \text{sign} A(i,j) $
all(V)	$\inf_i \text{sign} V(i) $
all(A,1)	idem que any
all(A,2)	idem que any

Scilab	
V	vecteur booléen ou scalaire
A	matrice booléenne ou scalaire
or(V)	%T si $V \neq 0$
or(A,1)	%T si $\text{col.}(j) \neq 0$
or(A,2)	%T si $\text{lig.}(i) \neq 0$
and(V)	%T si $\prod_i V(i) \neq 0$
and(A,1)	%T si $\prod_j A(:,j) \neq 0$
and(A,2)	%T si $\prod_i A(i,:) \neq 0$

```
// Scilab
A = [0,0,1.5;0,0.5,-1]
B = bool2s(A~=0)
V = or(A,1), W = and(A,1)
// V = [F,T,T], W = [F,F,T]
\columbreak
% MATLAB
```

```
A = [0,0,1.5;0,0.5,-1]
B = (A~=0), V = any(A,1), W = all(A,1)
% V = [0,1,1], W = [0,0,1]
```

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

On termine cette partie par un exemple un peu plus développé. Représenter une fonction $y = f(x)$ revient à se donner un vecteur d’abscisses $x = [x_1, x_2, \dots, x_n]$ et le vecteur d’ordonnées correspondant $y = [f(x_1), f(x_2), \dots, f(x_n)]$. 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)
\columbreak
```

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

`delta` est un vecteur-ligne booléen de même longueur que `x` valant 0 (ou %F) partout sauf à l’indice i tel que $x(i)=0$. On aurait pu écrire plus simplement `sin(x)/x`, mais on aurait obtenu une erreur à cet indice i . Le vecteur `delta` sert donc à corriger ce problème. Les fonctions `plot2d` et `plot` seront examinées plus tard.

Matrices spéciales

MATLAB et Scilab possèdent plusieurs outils de création automatique de matrices. Les plus évidents, mais aussi les plus utiles, sont `zeros`, `ones` et `eye`. La fonction `diag` permet aussi bien d’extraire la k -ième diagonale d’une matrice (sous forme d’un vecteur-colonne) que de construire des matrices à partir de ces diagonales.

MATLAB / Scilab	
<code>zeros(m,n)</code>	matrice $m \times n$ de 0
<code>ones(m,n)</code>	matrice $m \times n$ de 1
<code>eye(m,n)</code>	matrice Id $m \times n$

```
// Scilab
A = [1,2,3;4,5,6]
[nb_lig, nb_col] = size(A)
B = eye(nb_lig, nb_col)
B = eye(A), C = eye(size(A))
% MATLAB
A = [1,2,3;4,5,6]
[nb_lig, nb_col] = size(A)
B = eye(nb_lig, nb_col)
B = eye(size(A))
```

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

Dans la construction $A = \text{diag}(V, k)$, la matrice obtenue est carrée de dimension $\text{dim}(V) + k$ et a toutes ses entrées nulles sauf sur la k -ième diagonale qui est égale au vecteur (-colonne) V .

```
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)
```

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

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)	depuis la diagonale k
tril(A)	triangulaire inférieure
tril(A,k)	depuis la diagonale k

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

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

Opérations avancées

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

MATLAB	Scilab
A	matrice $m \times n$
V	vecteur-ligne $1 \times n$
W	vecteur-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)	produit par colonne
W = prod(A,2)	produit par ligne
cumsum(V)	sommes cumulées
cumsum(A,1)	somme par colonne
cumsum(A,2)	somme par ligne
cumprod(V)	produits cumulés
cumprod(A,1)	produits par colonne
cumprod(A,2)	produits par ligne

```
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)
```

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} \quad B = [9 \ 12 \ 15 \ 18] \quad C = \begin{bmatrix} 10 \\ 14 \\ 30 \end{bmatrix}$$

$$D = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 3 & 5 & 7 & 9 \\ 9 & 12 & 15 & 18 \end{bmatrix} \quad E = \begin{bmatrix} 1 & 3 & 6 & 10 \\ 2 & 5 & 9 & 14 \\ 6 & 13 & 21 & 30 \end{bmatrix}$$

$$F = [12 \ 42 \ 96 \ 180] \quad G = \begin{bmatrix} 1 & 2 & 6 & 24 \\ 2 & 6 & 24 & 120 \\ 6 & 42 & 336 & 3024 \end{bmatrix}$$

Toujours pour éviter d'écrire des petits programmes élémentaires, mais très lents, on dispose aussi des fonctions `min` et `max`. Pour des vecteurs, MATLAB et Scilab opèrent de la même manière, pour des matrices, MATLAB et Scilab diffèrent sur la syntaxe. Dans ce qui suit, `A`, `B`, `M` sont des matrices $m \times n$, `M1`, `K1` sont des vecteurs-ligne, `M2`, `K2` sont des vecteurs-colonne, `V` est un vecteur-ligne ou -colonne, `m`, `k` sont des scalaires. `M1(j) = mini A(i, j)` désigne le minimum dans chaque colonne, c'est-à-dire le minimum suivant le premier indice, `M2(i) = minj 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	Scilab	Remarques
<code>m = min(V)</code>	<code>m = min(V)</code>	$m = \min_i V(i)$
<code>[m,k] = min(V)</code>	<code>[m,k] = min(V)</code>	$m = V(k)$
<code>m = max(V)</code>	<code>m = max(V)</code>	$m = \max_i V(i)$
<code>[m,k] = max(V)</code>	<code>[m,k] = max(V)</code>	$m = V(k)$
<code>M = min(A,B)</code>	<code>M1 = min(A,1)</code>	$M(i, j) = \min\{A(i, j), B(i, j)\}$
<code>[M1,K1] = min(A, [], 1)</code>	<code>[M1,K1] = min(A, 'r')</code>	$M1(j) = \min_i A(i, j)$
<code>[M2,K2] = min(A, [], 2)</code>	<code>[M2,K2] = min(A, 'c')</code>	$M2(i) = \min_j A(i, j)$

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

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

$$M2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, K2 = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$M1 = [1 \ 2 \ 1], K1 = [1 \ 1 \ 2]$$

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, \dots (toujours ligne en Scilab et de même type que `V` en MATLAB) tel que $V(i_1) \neq 0, V(i_2) \neq 0, \dots$ et $V(i) = 0$ partout ailleurs. Si `A` est une matrice, `find(A)` retourne deux vecteurs (i_1, i_2, \dots) et (j_1, j_2, \dots) tels que $A(i_1, j_1) \neq 0, A(i_2, j_2) \neq 0, \dots$ et $A(i, j) = 0$ partout ailleurs.

MATLAB / Scilab	
<code>V</code>	ligne / colonne
<code>A</code>	matrice
<code>II, JJ</code>	ligne / colonne
<code>II = find(V)</code>	$V(i) \neq 0$
<code>[II, JJ] = find(A)</code>	$A(i, j) \neq 0$

`[II, JJ] = find(A)`

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

`JJ = [1. 1. 2. 2. 2. 3. 3.],`

`II = [1. 2. 1. 2. 3. 2. 3.]`

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

Bien que l'outil soit un peu réservé aux problèmes statistiques, on trouve dans la configuration de base des fonctions, appelées générateurs aléatoires de nombres, qui simulent un échantillon d'une loi donnée. Les deux générateurs de base sont la loi uniforme sur $[0, 1]$ et la loi normale sur \mathbb{R} , centrée et réduite. Là encore, MATLAB et Scilab diffèrent. Dans ce qui suit, k, m, n sont des entiers et `A` est une matrice de dimension $m \times n$.

MATLAB	Scilab	Remarques
<code>A = rand(m,n)</code>	<code>rand(m,n,'uniform')</code>	loi uniforme
<code>A = randn(m,n)</code>	<code>rand(m,n,'normal')</code>	loi normale
<code>rand('state',k)</code>	<code>rand('seed',k)</code>	amorce initiale

Voici comment modifier l'amorce (ou état) du générateur à chaque appel de `rand` ou `randn` :

```
// Scilab
format('v',25)
rand('seed',getdate('s'))
W = rand(1,5,'normal')
```

```
% MATLAB
format('long')
rand('state',100*sum(clock))
V = rand(1,5)
```

MATLAB et Scilab proposent tous les deux des fonctions de tri, de moyenne et d'écart type. Ils 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 `j` et une matrice `C` formée de vecteurs-colonne indiquant la permutation qui assure que l'on ait l'identité $C(B(:,j),j) = A(:,j)$. Dans ce qui suit, `X, Y, Z` sont des vecteurs et `A, B, C` sont des matrices de même dimension.

MATLAB	Scilab	Remarques
<code>[X,Y] = sort(Z)</code>	<code>[X,Y] = sort(Z)</code>	tri d'un vecteur
<code>[A,B] = sort(C,1)</code>	<code>[A,B] = sort(C,'r')</code>	tri sur indice de ligne
<code>[A,B] = sort(C,2)</code>	<code>[A,B] = sort(C,'c')</code>	tri sur indice de colonne

```
//Scilab
C = [9, 4, 8, 2, 4;...
     1, 9, 5, 2, 6;...
     7, 6, 4, 7, 2]
[A,B] = sort(C)
%MATLAB
[A,B] = sort(C,1)
```

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

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

Les fonctions statistiques usuelles, comme la moyenne, la médiane ou l'écart type, ont été implémentées dans les versions de base. Dans ce qui suit, X est une matrice $m \times n$, A est un vecteur-ligne de dimension n et B est un vecteur-colonne de dimension m .

MATLAB	Scilab	Remarques
<code>A = mean(X,1)</code>	<code>A = mean(X,'r')</code>	moyenne par indice de ligne
<code>B = mean(X,2)</code>	<code>B = mean(X,'c')</code>	moyenne par indice de colonne
<code>A = median(X,1)</code>	<code>A = median(X,'r')</code>	médiane par indice de ligne
<code>B = median(X,2)</code>	<code>B = median(X,'c')</code>	médiane par indice de colonne
<code>A = std(X,1)</code>	<code>A = st_deviation(X,1)</code>	écart type par indice de ligne
<code>B=std(X,2)</code>	<code>B=st_deviation(X,2)</code>	écart type par indice de colonne

Pour le calcul de l'écart type σ , MATLAB et Scilab utilisent des fonctions différentes. Néanmoins, dans les deux cas, le coefficient de normalisation est $N - 1$, où N est la taille des données. Dans l'exemple suivant, on modélise une loi binomiale de paramètres $n = 4$ et $p = \frac{1}{3}$. Les valeurs $PP0 \dots PP4$ représentent les fréquences empiriques avec lesquelles on obtient $0, 1, \dots, 4$. Les valeurs $pp0 \dots pp4$ représentent les probabilités théoriques.

```
//Scilab
format('v',6); pp = 1/3;
xx = bool2s(rand(4,10000,'uniform')<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
0.193	0.387	0.31	0.096	0.015

Probabilités théoriques :

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

1.6. Graphiques en 2D

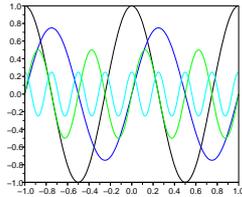
MATLAB et Scilab ont deux approches très différentes pour représenter des graphiques. Il existe néanmoins une approche minimaliste commune aux deux logiciels. Dans ce qui suit, X et Y sont des vecteurs ou des matrices. Les fonctions graphiques de base sont `plot` pour MATLAB et `plot2d` pour Scilab.

MATLAB	Scilab	Remarques
<code>clf()</code>	<code>clf()</code>	effacement du graphique courant
<code>plot(X,Y)</code>	<code>plot2d(X,Y)</code>	représentation de Y en fonction de X
<code>hold on</code>	<code>set(gca(),"auto_clear","off")</code>	surimpression des graphiques
<code>hold off</code>	<code>set(gca(),"auto_clear","on")</code>	un graphique à chaque appel de <code>plot</code>
<code>subplot</code>	<code>subplot</code>	partition de la fenêtre

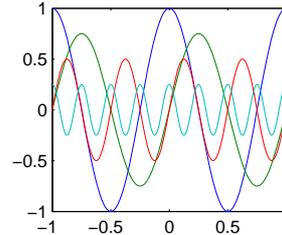
Il y a plusieurs cas d'utilisation :

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

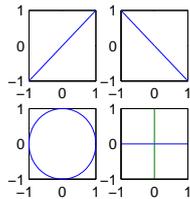
```
//Scilab
clf();
xx = (-1):0.01:1; yy = cos(2*pi*xx);
zz = 0.75*sin(2*pi*xx);
ww = 0.5*sin(4*pi*xx);
vv = 0.25*cos(8*pi*xx);
plot2d(xx', [yy', zz', ww', vv']');
```



```
%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']');
```



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.



```
%MATLAB (pour Scilab : pi -> %pi)
xx = pi*(-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(xx),sin(xx));
subplot(2,2,4);
plot([[ -1;1], [0;0]], [[0;0], [-1;1]]);
```

Graphique sous MATLAB

Si l'on ne précise rien, la fonction `plot(x,y)` trace un graphique en ligne continue (brisée si les points sont espacés) et noire. Chaque appel de `plot` efface l'ancien graphique. Il est possible néanmoins de changer le style et la couleur du tracé ainsi que la forme des points.

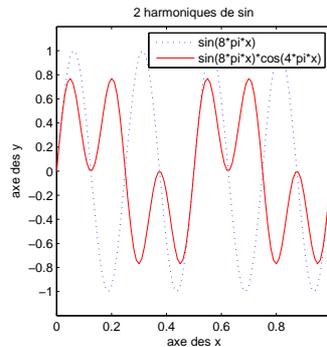
Style du tracé	
-	trait continu
:	pointillé
-.	trait/point
-	trait/trait
<i>rien</i>	pas de trait

Forme des points	
.	point
o	cercle
x	la lettre x
+	plus
*	étoile
s	carré
d	losange

Couleur du tracé	
b	"blue" [bleu]
g	"green" [vert]
r	"red" [rouge]
c	"cyan" [cyan]
m	"magenta" [magenta]
y	"yellow" [jaune]
k	"black" [noir]

La forme générale d'un tracé d'une ou plusieurs courbes est donnée par

```
%MATLAB
x = 0:0.01:1;
y = sin(2*pi*4*x); z = y.*cos(2*pi*2*x);
plot(x,y,'b:',x,z,'r-');
axis([0,1,-1.2,1.2]);
title('2 harmoniques de sin');
legend('sin(8*pi*x)',...
       'sin(8*pi*x)*cos(4*pi*x)');
xlabel('axe des x'); ylabel('axe des y');
```



Un graphique peut être enrichi par du texte ou avoir une apparence différente, comme dans la figure précédente. Les commandes les plus courantes sont données dans les deux tableaux suivants :

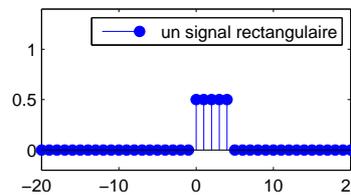
Textes et légende	
legend	légende des courbes
title	titre du graphique
xlabel	texte sur l'axe des x
ylabel	texte sur l'axe des y
text	texte positionné

Scilab : axes et cadre	
axis([xmin,xmax,ymin,ymax])	bornes des axes
set(gca,'XTick',[x1,x2,...])	taquets numériques
set(gca,'YTick',[y1,y2,...])	taquets numériques
set(gca,'XTickLabel',{'a',...})	taquets alphabétiques
grid on ; grid off ;	quadrillage

La fonction plot admet plusieurs variantes.

Autres versions de plot	
loglog	logarithmique en x/y
semilogx	logarithmique en x
semilogy	logarithmique en y
stem	graphique en peigne
bar	graphique en barre
stairs	marches d'escalier
hist	histogramme

```
%MATLAB
x = (-20):(20); y = ((x >= 0)&(x<5))/2;
stem(x,y,'filled');
axis([-20,20,-0.2,1.4]);
legend('un signal rectangulaire');
```



Graphique sous Scilab

Pour connaître toutes les fonctionnalités graphiques de Scilab, le plus simple est de les obtenir en ligne par `help Graphics`. On peut néanmoins se contenter des informations suivantes dans un premier temps. Par défaut, l'utilisation successive de `plot2d` superpose les graphiques les uns sur les autres. Pour effacer un graphique précédent, on utilise `clf()`. Comme pour MATLAB, la fonction `plot2d` possède plusieurs variantes. De plus, les fonctionnalités de `plot2d()`, ainsi que des versions `plot2dx()`, peuvent être modifiées par un ensemble d'options facultatives sous la forme `plot2d(X,Y,options)` où `option = clef1=valeur1,clef2=valeur2`, etc.

Scilab	
plot2d1	voir option de plot2d
plot2d2	marches d'escalier
plot2d3	barres verticales
plot2d4	lignes de courant
histplot	histogramme

Clef	Valeur
style	[i,j,...]
rect	[xmin,ymin,xmax,ymax]
logflag	"nn" "nl" "ln" "ll"
frameflag	f = 0 : 8
axesflag	a = 0 : 5
nax	[nx,Nx,ny,Ny]
leg	"leg1@leg2@..."

– `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, l'indice j, la deuxième courbe et ainsi de suite. Si l'indice est positif, la courbe est dessinée en trait plein d'une couleur donnée par `getcolor()`. Si l'indice est négatif, la courbe est dessinée point par point en utilisant un type de marqueur donné par `getmark()`.

style : couleur des courbes					
1	"black"	[0,0,0]	6	"magenta"	[255,0,255]
2	"blue"	[0,0,255]	7	"yellow"	[255,255,0]
3	"green"	[0,255,0]	8	"white" [blanc]	[255,255,255]
4	"cyan"	[0,255,255]
5	"red"	[255,0,0]	32	"gold" [or]	[255,215,0]

mark_style :			
0	.	-5	◇
-1	+	-6	△
-2	×	-7	▽
-3	⊕	-8	⊠
-4	◆	-9	○

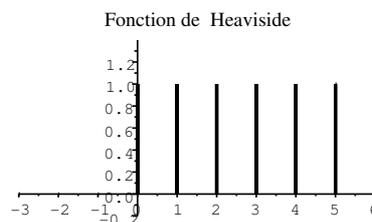
- **rect** : un vecteur-ligne à 4 entrées donnant les bornes du dessin (les bornes horizontales dans `[xmin,xmax]`, les bornes verticales dans `[ymin,ymax]`).
- **logflag** : choix d'une échelle linéaire `n` ou logarithmique `l` suivant les directions horizontales (première lettre) ou verticales (seconde lettre).
- **frameflag** : cette option permet de calculer automatiquement les bornes du dessin si l'option **rect** n'est pas utilisée. Les différentes possibilités sont données dans le tableau suivant. Dans l'option isométrique, les axes horizontaux et verticaux ont la même échelle. Pour les trois dernières options, tous les graphiques précédents sont redessinés suivant les dernières valeurs de `plot2d`.

frameflag : bornes du dessin	
f = 0	valeurs du graphique précédent
f = 1	valeurs données par rect
f = 2	valeurs utilisant min/max de x/y
f = 3	échelle isométrique sur rect
f = 4	échelle isométrique sur min/max
f = 5	idem à f = 1 en mieux dessiné
f = 6	idem à f = 2 en mieux dessiné
f = 7	idem à f = 1, redessiné
f = 8	idem à f = 2, redessiné
f = 9	idem à f = 2 et f = 8 (par défaut)

axesflag : placements des axes	
a = 0	aucun axe + aucun cadre
a = 1	axes des xy en bas à gauche + un cadre
a = 2	aucun axe + un cadre
a = 3	axes des xy en bas à droite + aucun cadre
a = 4	axes des xy centrés + aucun cadre
a = 5	axes des xy centrés en x = y = 0 + un cadre
a = 9	axes des xy en bas à gauche + aucun cadre

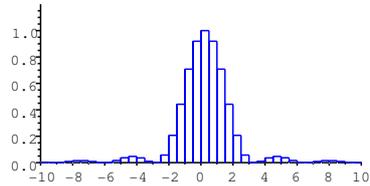
- **axesflag** : cette option permet de placer les axes et éventuellement un cadre autour du graphique. Les différentes options sont données dans le tableau précédent.
- **nax** : cette option permet de contrôler le nombre de graduations principales `Nx,Ny` et de graduations secondaires `nx,ny` entre deux graduations principales qui sont affichées sur les axes. Cette commande ne fonctionne qu'avec l'option **axesflag** = 1.
- **leg** : cette option permet d'afficher une légende à chaque courbe sous forme d'une chaîne de caractères "`leg1@leg2@ ...`" où "`leg1`" est la légende du premier graphique et ainsi de suite. Le graphe suivant illustre ces notions en représentant la fonction de Heaviside discrète et tronquée sur l'intervalle de temps `[-3, +5]`. L'intervalle de temps est représenté par le vecteur `xx = [-3,-2,-1,0,1,2,3,4,5]` et les valeurs de la fonction par `yy = [0,0,0,1,1,1,1,1,1]`.

```
// Scilab
xx = (-3):1:5; yy = bool2s(xx>=0);
plot2d3(xx',yy',style=1,...
        rect=[-3,-0.2,6,1.4],...
        frameflag=1,axesflag=5);
ff = get('current_figure');
aa = ff.children; aa.box = 'off';
aa.title.text = 'Fonction de Heaviside';
aa.title.font_size = 2;
aa.title.font_style = 2;
aa.children.children.thickness = 2;
```

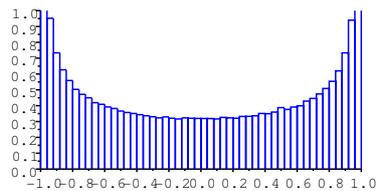


Dans un second exemple (premier graphe ci-dessous), on utilise simultanément `plot2d2` et `plot2d3` pour obtenir un diagramme en barre de la fonction $y = (\sin(x)/x)^2$. Noter l'artifice `dd = bool2s(xx==0)` qui évite de diviser par zéro.

```
//Scilab
xx = (-10):0.5:10; dd = bool2s(xx==0);
yy = ((sin(xx)+dd)./(xx+dd))^2;
plot2d2(xx',yy',style=2,...
        rect=[-10,0,10,1.2],frameflag=1);
plot2d3(xx',yy',style=2,...
        rect=[-10,0,10,1.2],frameflag=1);
```



```
//Scilab
stacksize(2000000); nn = 400000;
yy = zeros(nn,1); xx = 2*rand()-1;
nb = 50; yy = zeros(1,nn);
for ii = 1:nn,
    yy(ii)=xx; xx = 1-2*xx^2;
end
histplot(nb,yy,style=2,...
        rect=[-1,0,1,1],frameflag=1);
```



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

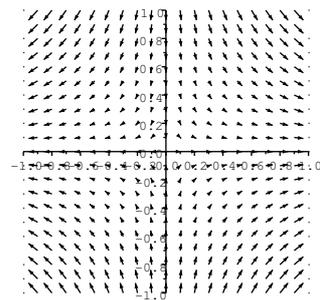
Fonctions 2D graphiques avancées

On peut représenter des données dans MATLAB et Scilab de plusieurs manières différentes. Le tableau suivant énumère quelques possibilités.

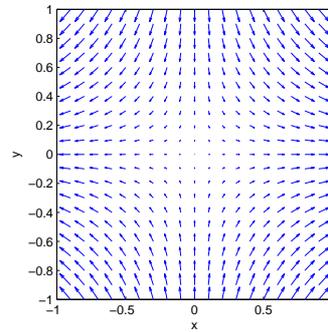
MATLAB	Scilab	Remarques
quiver	champ	champ de vecteurs
contourf	grayplot	courbes de niveaux lissées
contour	contour2d	courbes de niveaux
meshgrid	ndgrid	création d'un quadrillage

Le graphique suivant représente le champ de vecteurs autour d'un point fixe de type « selle ». L'équation différentielle est donnée par $\dot{x} = x$, $\dot{y} = -y$, l'unique point fixe est $x = y = 0$ et le champ de vecteurs est $u(x, y) = x$, $v(x, y) = -y$.

```
//Scilab
xmin = -1; xmax = 1; ymin = -1; ymax = 1;
cadre = [xmin,ymin,xmax,ymax];
xx = xmin:0.1:xmax; yy = ymin:0.1:ymax;
[XX,YY]=ndgrid(xx,yy); UU = XX; VV = -YY;
champ(xx,yy,UU,VV,rect=cadre);
aa = get('current_axes');
aa.x_location = 'middle';
aa.y_location = 'middle';
```

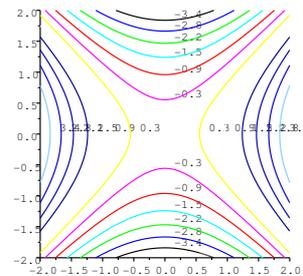


```
%MATLAB
xmin = -1; xmax = 1; ymin = -1; ymax = 1;
cadre = [xmin,ymin,xmax,ymax];
xx = xmin:0.1:xmax; yy = ymin:0.1:ymax;
[XX,YY] = meshgrid(xx,yy);
UU = XX; VV = -YY;
quiver(XX,YY,UU,VV,'b')
xlabel('x'); ylabel('y');
xlim([xmin,xmax]); ylim([ymin,ymax]);
```

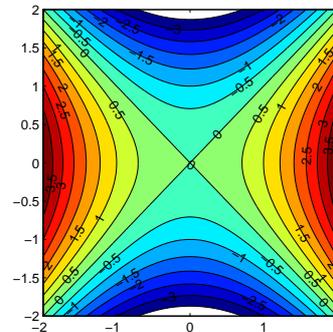


Les graphes suivants représentent 10 courbes de niveaux de la surface $z = x^2 - y^2$ définie sur $x \in [-2, 2]$ et $y \in [-2, 2]$. Le code MATLAB donne une version lissée des courbes de niveaux.

```
//Scilab
xx = (-2):0.01:(2);
yy = (-2):0.01:(2);
[XX,YY] = ndgrid(xx,yy);
ZZ = XX.^2-YY.^2;
nz = 12;
xset("fpf", "%.1f");
contour2d(xx,yy,ZZ,nz,style=1:nz,...
rect=[-2,-2,2,2],frameflag=3);
```



```
%MATLAB
xx = (-2):0.01:(2);
yy = (-2):0.01:(2);
[XX,YY] = meshgrid(xx,yy);
ZZ = XX.^2-YY.^2;
vz = (-3.5):0.5:3.5;
[C,h] = contourf(XX,YY,ZZ,vz);
clabel(C,h);
```



1.7. Programmation

MATLAB et Scilab possèdent toutes les structures de programmation d'un langage classique. Rappelons cependant qu'une approche vectorielle est préférable, pour ces logiciels de calcul scientifique, à une approche classique. Nous avons déjà vu comment écrire des fichiers `script` et comment les exécuter avec `exec('nom_fichier.sce')` pour Scilab, et `nom_fichier` seulement pour MATLAB. On peut aussi compiler un bloc de codes, sous forme de fonction, indépendamment du reste du programme principal, et réutiliser cette fonction partout dans le programme.

MATLAB	Scilab
<pre>function [Y1,Y2] = nom_fonction(X1,X2) ... end</pre>	<pre>function [Y1,Y2] = nom_fonction(X1,X2) ... endfunction</pre>

La première ligne du fichier doit impérativement contenir la déclaration de fonction, et le nom du fichier doit se terminer par `.m` pour MATLAB et éventuellement par `.sci` pour Scilab. Le nombre de variables en entrée `X1`, `X2`, ... et en sortie `Y1`, `Y2`, ... peut être quelconque et même nul comme dans `function nom_fonction(X1,...)`. Les variables passées en argument ne sont pas modifiées dans le programme qui les appelle, et toutes les variables déclarées à l'intérieur de la fonction sont locales à celle-ci. L'exécution de la fonction se fait en l'appelant par son nom, sans l'extension. Pour Scilab, un chargement avant exécution de la fonction est nécessaire; on l'effectue avec `exec('nom_fonction.sci')`.

<pre>//Scilab //declaration et sauvegarde par somme.sci function [Y]=somme(n) Y=sum(ones(1,n)); endfunction //chargement et execution exec('somme.sci'); tic(),somme(1000000),toc() // 0.062 sec</pre>	<pre>%MATLAB %declaration et sauvegarde par somme.m function [Y]=somme(n) Y=0; for ii=1:n, Y=Y+1; end end %execution (pas de chargement) tic(),somme(1000000),toc() % 0.012 sec</pre>
--	---

Comme tout logiciel de programmation, MATLAB et Scilab possèdent les structures usuelles de boucles.

MATLAB/Scilab
<pre>for variable = vecteur, instructions, end while expression, instructions, end if expression, instructions, else instructions, end switch selecteur, case valeur1, instructions, otherwise instructions, end</pre>

<pre>//Scilab n = 0; x = 0; while n<100, n=n+1; x=x+1/n^2; end x // 1.6349839 %pi^2/6 // 1.6449341</pre>	<pre>%MATLAB x = 0; format long for n=0:9999, n=n+1; if mod(n,2)~=0,x=x+1/n^2; end end x % 1.233650550136341 pi^2/8 % 1.233700550136170</pre>
---	---

Enfin, MATLAB et Scilab peuvent exporter et importer une très grande variété de fichiers, aussi bien sous format `texte` que sous un format graphique `.ps`, `.pdf`, ou bien sous un format audio `.au`, `.wav` ou video `.avi`.