Université Clermont 1
IUT Info

Adrien Wohrer
Année 2017-2018

Scilab : Manuel de référence

Scilab est utilisé dans les cours de mathématiques suivants : Algèbre linéaire (1A), Analyse (1A), Probabilités et Statistiques (2A). Ce document détaille les principales fonctionnalités que vous devrez maîtriser, sous forme de tableaux thématiques. **Conservez-en une copie dans votre espace personnel.**

1 Généralités

Scilab est un logiciel de calcul scientifique que nous utiliserons dans plusieurs cours de mathématiques lors des deux années à venir. En résumé, Scilab est une « grosse calculatrice », assortie d'un langage de programmation simple.

Il a été développé en France au sein de l'INRIA (Institut National de Recheche en Informatique et Automatique). Il constitue une alternative gratuite au célèbre logiciel payant **Matlab**. Les deux langages ont (presque) la même syntaxe et la même utilisation.

La console. Pour lancer Scilab, on tape la commande `scilab` dans un terminal. Une fenêtre Scilab s'ouvre et le prompt (–>) apparait dans la *console* Scilab. Vous pouvez ainsi définir des *variables* et travailler avec :

```
a = 5;
b = 2;
c = a*b
```

Si vous mettez un « **;** » à la fin de la ligne, le résultat ne s'affiche pas. Si vous ne mettez rien, le résultat s'affiche.

Le workspace. L'ensemble des variables existantes à un moment donné constitue ce qu'on appelle le *workspace* de Scilab.

Voici les commandes basiques pour travailler sous l'environnement Scilab.

L'environnement Scilab	
;	À la fin d'une ligne : supprime l'affichage du résultat
disp	Affiche sur la sortie standard
who_user	Affiche les variables créées dans le workspace
clear	Nettoie des variables du workspace
halt	Suspend provisoirement l'execution
tic	Lance un chronomètre
toc	Arrête un chronomètre (lancé plus haut avec **tic**)
pwd, **ls**, **cd**,...	Navigation dans le système de fichiers
help	Ouvre l'aide Scilab. Pour ouvrir l'aide sur une commande donnée : `help nom_de_la_commande`

Exemple : Le code ci-dessous illustre l'utilisation de quelques fonctions.

```
dollar = 5;                                    // définit la variable 'dollar' (sans l'afficher)
disp(dollar)                                   // affiche la valeur de la variable 'dollar'
disp("Bob a gagné "+string(dollar)+" dollars")          // affichage plus complexe
                // Remarque: string() convertit un nombre en une chaine de caractères

tic(); halt("Frappez une touche"); disp("Merci!"); t=toc();
     // Lance un chronomètre, attend une frappe utilisateur, stocke le temps écoulé
disp("Vous avez attendu pendant "+string(t)+" secondes")     // Affiche le résultat

who_user;                                      // variables utilisateur dans le workspace
clear dollar                                   // supprime la variable 'dollar'
who_user;                          // (nouvelles) variables utilisateur dans le workspace
```

2 Manipulation des tableaux

En Scilab, toutes les variables sont des tableaux. Par exemple :

- Un **nombre** est un tableau de taille 1×1.
- Un **vecteur colonne** est un tableau de taille $n \times 1$.
- Un **vecteur ligne** est un tableau de taille $1 \times n$.
- Une **matrice** est un tableau de taille $m \times n$.

Il existe donc tout un jeu de commandes spécifiques pour manipuler les tableaux.

Manipulation de tableaux	
[]	Construction manuelle d'un tableau
()	Accède au contenu du tableau
:	Opérateur spécial d'indexation (exemples ci-dessous)
$	Index du dernier élément d'un tableau
size	Dimensions d'un tableau
length	Nombre total d'éléments dans un tableau
isempty	Teste si un tableau est vide
isequal	Teste si deux tableaux sont identiques
sum	Somme des éléments d'un tableau
prod	Produit des éléments d'un tableau
zeros	Initialise un tableau rempli de 0
ones	Initialise un tableau rempli de 1
rand	Initialise un tableau rempli de nombres aléatoires

ATTENTION : EN SCILAB LES INDICES COMMENCENT À 1.

Exemple : Pour initialiser un petit tableau « à la main », on utilise des crochets **[ ]**. Puis on rentre le contenu *ligne par ligne*, en séparant les lignes par un « **;** ».

```
n = 5                                  // nombre = tableau de taille 1x1
size(n)                                // taille du ``tableau'' n
B = [2*3, 5, 7, 1/9, 0]                // vecteur ligne = tableau de taille 1x5
C = [5; 7; 11]                         // vecteur colonne = tableau de taille 3x1
A = [1 2 3 4; 5 6 7 8]                 // matrice = tableau de taille 2x4
```

Pour initialiser un grand tableau, il existe des fonctions spécifiques :

```
B = 1:10                               // tableau de taille 1x10, avec les entiers de 1 à 10
C = 3.4 : 0.1 : 5                      // tableau de taille 1x17, égal à [3.4, 3.5, 3.6, ... , 5]
D = ones(4,5,3)                        // tableau de taille 4x5x3, rempli de 1
```

Pour accéder au contenu d'un tableau, on utilise les parenthèses **( )** et l'opérateur « **:** ».

```
A( 1 , 3 )                             // élément de A à la ligne 1 et colonne 3
A( 2 , 2:4 )                           // sous-tableau donné par la ligne 2 et les colonnes 2 à 4
A( 2 , 3 ) = 13                        // modifie le contenu de l'élément à la ligne 2 et colonne 3
A( : , 4 ) = [9; 10]                   // modifie le contenu de la 4ème colonne
```

3 Mathématiques

Toutes les opérations mathématiques de base peuvent être appliquées à des nombres, mais aussi directement à des tableaux. C'est ça qui fait toute la puissance de Scilab ! Dans ce cas :

- L'opération en question est réalisée élément par élément.
- Le résultat est lui-même un tableau, de même taille que le(s) tableau(x) en entrée.

Attention ! si vous calculez $A+B$ pour deux tableaux A et B, ils doivent nécessairement avoir la même taille, sinon Scilab renverra une erreur. (Et de même pour toute autre opération.)

Opérateurs mathématiques	
A +B	Addition
A - B	Soustraction
A .*B	Multiplication (élément par élément)
A ./B	Division (élément par élément)
A .^B	Puissance (élément par élément)

Attention ! Les opérations multiplication, division et puissance doivent être précédées d'un point. Sinon, Scilab les interprète comme des multiplications matricielles (voir plus loin).

Opérateurs booléens	
%t	Booléen avec la valeur « vrai »
%f	Booléen avec la valeur « faux »
A == B	Teste si $A = B$ (élément par élément)
A ~= B ou **A <> B**	Teste si A est différent de B (élément par élément)
A < B	Teste si A est inférieur à B (élément par élément)
A >= B	Teste si A est supérieur ou égal à B (élément par élément)
A | B	Teste si A OU B est vrai (élément par élément)
A & B	Teste si A ET B sont vrais (élément par élément)
~A	Négation de A (élément par élément)
find	Trouve tous les indices vrais dans un vecteur booléen

Rappel : Les tests booléens peuvent aussi prendre en entrée des tableaux ; auquel cas le test est effectué, de manière indépendante, pour chaque élément du tableau.

Exemple : Trouver tous les indices où deux vecteurs (de même taille !) sont égaux :

```
A = 0:2:6                                  // renvoie  A = [0 2 4 6]
B = 3*[0 1 1 2]                            // renvoie  B = [0 3 3 6]
C = (A==B)                          // renvoie  C = [%t %f %f %t]
find(A==B)                                 // renvoie [1 4]
```

Fonctions mathématiques usuelles	
%pi	Nombre π
%e	Nombre e
%i	Nombre i (unité imaginaire)
abs	Valeur absolue (ou module pour les nombres imaginaires)
sqrt	Racine carrée
exp	Fonction exponentielle
log, **log10**	Logarithme néperien, logarithme décimal
cos, **sin**, **tan**	Cosinus, sinus, tangente
acos, **asin**, **atan**	Réciproques des précédentes
round, **ceil**, **floor**	Arrondi à l'entier (différentes versions)

Rappel : Les fonctions peuvent être appelées directement sur des tableaux.

Scilab est originellement un langage de calcul matriciel. Ses variables de base sont des tableaux à deux dimensions—en d'autres termes des *matrices*. Et la multiplication « par défaut » de Scilab est la multiplication matricielle.

Voici une liste de quelques commandes relatives au calcul matriciel. À la fin du cours d'algèbre linéaire, vous saurez la signification de chacune d'entre elles. :)

Opérateurs et fonctions matricielles	
A*B	Produit matriciel AB
A'	Transposée de la matrice A
A^n	Puissance matricielle A^n
inv(A) ou **A^-1**	Inverse de la matrice A
A/B	Division matricielle à droite AB^{-1}
A\B	Division matricielle à gauche $A^{-1}B$
eye(n,n)	Matrice identité de taille n
diag	Manipulation de matrices diagonales
det(A)	Déterminant de la matrice A
trace(A)	Trace de la matrice A
spec(A)	Décomposition en valeurs propres de la matrice A
Matplot(A)	Représentation graphique du contenu de la matrice A

Attention ! Notez l'absence de point devant les opérations de multiplication, indiquant qu'il s'agit du produit matriciel—qui n'est PAS le produit élément par élément. Donc **pensez au point** si vous voulez effectuer un produit « classique » entre deux tableaux !!!

4 Graphiques

Une des raisons qui nous a poussés à travailler avec Scilab est sa gestion des graphiques, à la fois simple et riche. Pour un aperçu rapide de toutes les possibilités graphiques, tapez

```
help Graphics
```

C'est le meilleur moyen de trouver la fonction dont vous avez besoin ! Dans ce manuel, nous n'exposons que le strict nécessaire pour démarrer.

Gestion générale des objets graphiques	
f=figure(*options***)**	Crée une figure (qui devient la figure courante)
scf(f)	Fait de **f** la figure courante
xtitle	Donne un titre à la figure courante (ainsi qu'aux axes)
clf, clf(f)	Efface la figure courante, ou la figure **f**
close, close(f)	Ferme la figure courante, ou la figure **f**
xdel(winsid())	Ferme toutes les figures
drawlater()	Utile pour faire des animations

Tracer des courbes	
x=linspace(a,b,N)	Discrétisation de l'intervalle $[a,b]$ en N points
plot(x,y,*options***)**	Trace la courbe d'abscisses **x** et d'ordonnées **y**
plot2d2	Comme **plot**, mais avec une courbe en escalier
param3d	Comme **plot**, mais avec une courbe en 3 dimensions
help LineSpec	Aspect des lignes (couleur, épaisseur, pointillés, etc)
help axes_properties	Plusieurs propriétés utiles se règlent à ce niveau : • `data_bounds` : changer les bornes de représentation • `grid` : tracer une grille derrière la figure, etc.

Exemple : Tracer deux courbes simples, puis modifier les bornes de représentation.

```
xdel(winsid());                                  // ferme toutes les figures préexistantes
x1 = linspace(-%pi, %pi, 50);                    // discrétisation de l'intervalle [-pi pi]
y1 = cos(3*x1);       // applique la fonction 'cos(3x)' à chaque élément du vecteur x1
x2 = -5 : .05 : 3 ;                              // discrétisation de l'intervalle [-5,3]
y2 = exp(x2)-x2;    // applique la fonction 'exp(x)-x' à chaque élément du vecteur x2
figure();                                        // ouvre une nouvelle figure
plot(x1, y1, "b", x2, y2, "--r", "LineWidth", 3) ;
            // trace la courbe (x1,y1) en bleu, et la courbe (x2,y2) en pointillés rouges
xtitle("Mes premières courbes","mes abscisses","mes ordonnées");          //titres
```

Vous pouvez ensuite modifier les bornes de représentation :

```
a = gca();                                       // axes de la figures courante
a.data_bounds = [-6 0; 2 4];                     // montre 'x' de -6 à 2, et 'y' de 0 à 4
a.grid = [1 1];                                  // ajoute une grille derrière les courbes
```

5 Programmer en Scilab

Scilab est également un langage de programmation, avec sa syntaxe propre et tous les éléments requis pour faire de l'algorithmique. L'utilisation est très proche du C, avec toutefois une syntaxe légèrement différente.

Bases d'algorithmique	
for ...end	Boucle "for"
while ...end	Boucle "while"
break, **continue**	Interrompre une boucle
if ...then ...else ...end	Instructions conditionnelles
select ...case ...case ...end	Instructions conditionnelles (autre forme)
function ...endfunction	Définir une fonction
argn	Compter les arguments lors de l'appel d'une fonction
return	Sortir d'une fonction (avant la fin)

Exemple : Voici une fonction qui calcule la somme des entiers de `n_debut` jusqu'à `n_fin`. Elle illustre la majorité des choses à connaître pour programmer en Scilab !

```
// Cette fonction se nomme 'calcule_somme'.
// Elle prend deux entrées 'n_fin' et 'n_debut'. Elle doit renvoyer une variable 'S'.
// Attention: si la fonction ne crée pas la variable 'S', Scilab retournera une erreur.

function S = calcule_somme(n_fin, n_debut)
    S = 0;                                          // Crée tout de suite la variable 'S'.
    // 1. VÉRIFICATION des entrées.
    // 'argn(2)' donne le nombre d'arguments en entrée
    if argn(2) == 1 then                    // L'utilisateur a donné un seul argument
        n_debut = 1;                  // On utilise une valeur par défaut : n_debut = 1
    elseif argn(2) == 0 then              // L'utilisateur n'a donné aucun argument
        disp("Merci de fournir au moins un argument en entrée.")
        return                                                  // Quitte la fonction.
    end
    // 2. CALCUL de S.
    for i = n_debut : n_fin                  // Syntaxe classique: for i = debut:pas:fin
        S = S + i ;
    end
endfunction                       // Ok : quand on quitte la fonction, 'S' existe toujours.
```

Vous pouvez ensuite appeler la fonction de la sorte :

```
ma_somme = calcule_somme(48, 12)                  // Somme des entiers de 12 à 48
ma_somme = calcule_somme(100)                     // Somme des entiers de 1 à 100
ma_somme = calcule_somme()                            // renvoie 0, et pas content
```

Fichiers de script. Dans la console, il suffit de taper une commande pour obtenir un résultat. C'est très pratique pour des commandes simples. Mais ça devient vite problématique pour des commandes plus compliquées, des instructions sur plusieurs lignes, etc.

Il devient alors nécessaire d'écrire un *script*, c'est-à-dire un fichier d'instructions que Scilab est capable d'exécuter. Plus le code est gros, plus il faudra faire attention à bien l'organiser, dans plusieurs fichiers séparés.

Organisation du code	
Fichier *.sce*	Script d'instructions (peut aussi contenir des fonctions)
Fichier *.sci*	Fichier contenant uniquement des fonctions
exec	Exécuter un fichier Scilab (*.sci* ou *.sce*)
pause	Insérer un point de déboguage dans un fichier Scilab
save, **load**	Sauver ou charger des données (en format binaire)

Exemple : Voici comment organiser idéalement le code de la page précédente.

1. Créez un premier fichier, intitulé *mes_fonctions.sci*. Recopiez dedans tout le code de la fonction `calcule_somme`, puis sauvez-le dans le répertoire courant de Scilab.
2. Créez un second fichier, intitulé *mon_script.sce*. Recopiez dedans les lignes suivantes :

   ```
   exec("mes_fonctions.sci", -1)          // Récupère la fonction 'calcule_somme'
   ma_somme = calcule_somme(48, 12)             // Premier appel de la fonction
   ma_somme = calcule_somme(100)              // Deuxième appel de la fonction
   disp("Fin du script, merci de votre attention.")
   ```

   Puis sauvez-le dans le répertoire courant de Scilab.
3. Enfin, dans la console Scilab, exécutez le script :

   ```
   exec("mon_script.sce", 0)
   ```

   Le second argument (ici, 0) définit le niveau de verbosité de l'exécution. Si vous le passez à -1, les lignes du script ne s'afficheront plus, même si elles n'ont pas de « **;** ».

Il s'agit là de la version la plus « propre » d'organisation. On sépare les fonctions dans des fichiers à part *.sci*, et les instructions dans un script *.sce*. Enfin, on appelle le fichier de script dans la console.

En pratique, sur du petit code, il est souvent préférable de mettre **les fonctions directement dans le script** *.sce* – ainsi, on travaille avec un unique fichier.

6 Probabilités et statistiques

Je liste ici quelques fonctions Scilab spécifiquement utiles pour le cours de proba/stats. Cette liste sert uniquement de référence ; vous devrez apprendre à utiliser ces fonctions en vous servant de la documentation.

Probabilités et Statistiques	
grand(m,n,*options***)**	Génère un tableau de nombres aléatoires, de taille (m,n). On peut tirer les nombres suivant plusieurs lois possibles : **"uin"** : loi uniforme sur des nombres entiers **"unf"** : loi uniforme sur des nombres réels **"bin"** : loi de Bernoulli ($N=1$) ou binômiale ($N>1$) **"nor"** : loi normale (Gaussienne) et encore plein d'autres choix possibles (voir la doc).
factorial	Factorielle d'un entier
bar(x,y,*options***)**	Diagramme en bâtons (usage similaire à la fonction **plot**)
histplot(ax,val,*options***)**	Histogramme des valeurs **val**, dans l'axe discrétisé **ax**
histc(ax,val)	Idem, mais calcule l'histogramme sans l'afficher
cdfnor	Fonction de distribution de la loi normale

Remarque : dans les versions anciennes de Scilab, la fonction **histc** n'existe pas ; elle sera alors remplacée par une fonction « maison » équivalente.

Exemple : Générer 100 lancers de dé, et représenter le résultat dans un histogramme.

```
res = grand(100,1,"uin",1,6)                  // 100 entiers tirés uniformément dans {1...6}
histplot(0:6, res);                                  // Histogramme des résultats (figure)
count = histc(6, res, %f)                          // Histogramme (calcul sans figure)
[m,k] = max(count)                                          // Chiffre sorti le plus souvent
disp("Le chiffre "+string(k)+" est sorti "+string(m)+ " fois.");
```

7 Autres sources d'information

On trouve une bonne documentation sur le site oficiel :

http ://www.scilab.org/resources/documentation

En particulier, les deux documents suivants sont disponibles sur l'ENT :

- Le tutoriel officiel **Scilab_debutant.pdf**, une bonne alternative à ce manuel.
- Pour aller plus loin, l'introduction officielle **introscilab.pdf** (en anglais).