

# Chapitre 4

## Scripts et fonctions

La façon dont Scilab a été utilisé dans les chapitres précédents peut donner l'impression qu'il s'agit simplement d'une "calculatrice améliorée" uniquement capable d'exécuter des commandes rentrées au clavier et de donner le résultat. En fait, Scilab est capable d'exécuter une suite de commandes stockées dans un fichier, et on préférera ce mode de fonctionnement dès que le nombre de lignes de commande est suffisamment grand (par exemple supérieur à 5 lignes). Cela permet aussi de ne pas avoir à retaper toute la suite de commandes si l'on a envie de changer la valeur de tel ou tel paramètre.

Les scripts et les fonctions sont des fichiers texte ordinaires créés à l'aide de l'éditeur de texte :

- Un *script* permet d'exécuter automatiquement une longue suite de commandes amenée à être répétée.
- Une *fonction* permet d'étendre la bibliothèque de fonctions standard de Scilab. La puissance de Scilab tient surtout à ce dernier aspect.

### 4.1 Les scripts

Lorsqu'un script est exécuté (nous verrons plus bas comment déclencher cette exécution), Scilab interprète tout simplement les commandes les unes après les autres comme si elles avaient été tapées au clavier. Cela veut donc dire que les variables créées et utilisées dans le script sont des variables de l'espace de travail de Scilab.

Voici un premier exemple : nous allons créer un script qui calcule quelques termes de la suite de Fibonacci dont la définition est la suivante :

$$\begin{cases} u_0 = 1, \\ u_1 = 1, \\ u_{k+2} = u_{k+1} + u_k, \quad k \geq 0 \end{cases}$$

Pour cela, sélectionner `Editeur` dans le menu de Scilab. Le fichier créé a par défaut le nom `Sans Titre` . Pour le changer ce nom il suffit de sélectionner `Enregister`

sous dans le menu fichier et de choisir le nom `fibonacci.sce`. La fenêtre créée doit ressembler à celle qui se trouve sur la figure 4.1. Vous pouvez maintenant saisir les

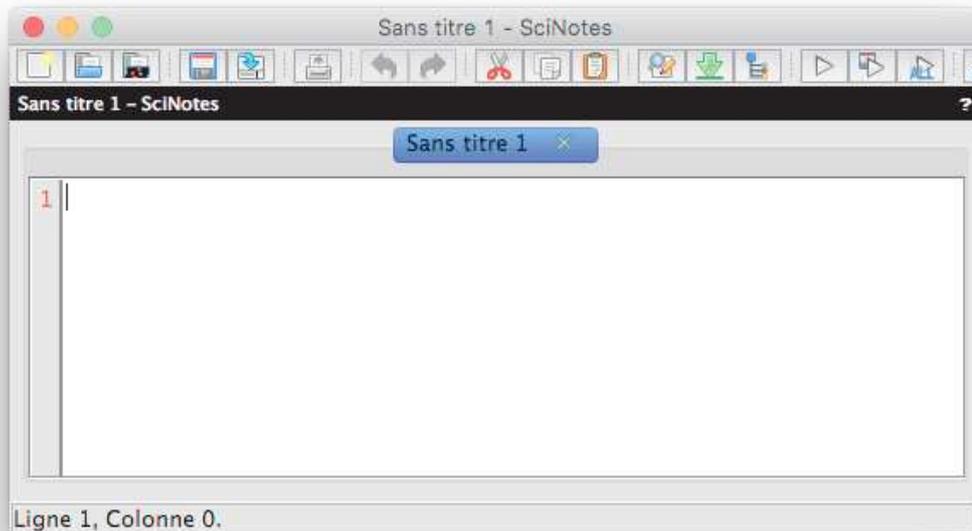


FIGURE 4.1 – Fenêtre de l’éditeur de texte

lignes suivantes dans la fenêtre de l’éditeur :

```
// Un script pour calculer les termes de la suite de Fibonacci
n=10;
u=zeros(n,1);
u(1:2)=[1;1];

for i=1:n-2
    u(i+2)=u(i+1)+u(i);
end

disp(u)
```

Les caractères `//` permet de spécifier que ce qui le suit ne doit pas être interprété par Scilab. Cela permet d’insérer des commentaires.

Nous verrons en détail comment utiliser l’instruction `for` un peu plus tard. Une fois que le texte est tapé dans la fenêtre, sauvez le fichier (utilisez le bouton .

Pour exécuter le script `fibonacci.sce`, il suffit de cliquer sur le bouton , ce qui devrait afficher les lignes suivantes dans la fenêtre de commande :

```
u =
```

```
1
1
2
3
5
8
13
21
34
55
```

## 4.2 Les fonctions

Un fichier contenant le mot *fonction* au début de la première ligne est une *fonction*. Une *fonction* diffère d'un *script* dans le sens où l'on peut lui transmettre des paramètres en entrée, et où les variables manipulées à l'intérieur sont *locales* à la fonction : cela veut dire que le seul lien avec des variables de l'espace de travail ne peut se faire qu'en passant des paramètres ou *arguments*. Les fonctions sont utiles pour étendre facilement les possibilités de Scilab.

A titre d'exemple, nous allons créer une fonction qui n'existe pas dans Scilab qui est une fonction calculant la factorielle. Vous pouvez donc créer un fichier nommé `fact.sci` et double-cliquer sur son icône pour invoquer l'éditeur de texte. Tapez dans la fenêtre le texte suivant

```
function [f]=fact(n)

// Cette fonction calcule la factorielle d'un nombre entier

if (n-floor(n)~ 0) | n<0
    error("erreur dans fact : l'argument doit être entier");
end

if n==0
    f=1;
else
    f=prod(1:n);
end

endfunction
```

Quelques explications sont nécessaires sur les lignes de cette fonction :

- La ligne `if (n - floor(n) ~= 0 ...` permet de tester si le nombre passé en argument est un entier naturel. La structure `if ... end` est assez classique et fonctionne presque comme dans le langage C.
- La commande `error` permet de stopper l'exécution de la fonction tout en renvoyant un message d'erreur à l'utilisateur.
- Les lignes suivantes montrent un exemple d'utilisation de la structure `if ... else ... end`.
- La fonction `prod` permet de calculer le produit des éléments du vecteur passé en argument.

Une fois que le texte est tapé dans la fenêtre, sauvez le fichier. Pour que Scilab puisse utiliser cette nouvelle fonction, vous devez tout d'abord exécuter le fichier grâce au bouton 

Il est possible de définir plus d'une fonction dans un seul fichier. Dans ce cas, toutes les fonctions sont définies dans Scilab au moment de l'exécution.

L'utilisation de la fonction `fact` peut se faire ensuite des deux façon suivantes :

```
--> fact(5)

ans =

    120.

--> p=fact(7);
```

Il est aussi possible de définir une fonction « en ligne », c'est à dire directement à partir de la ligne de commande de Scilab. Cela est pratique quand la fonction est très courte à écrire. Par exemple :

```
--> deff("c=plus(a,b)","c=a+b");

--> plus(1,2)

ans =

    3.
```

## 4.3 Contrôle d'exécution

La structure `if ... else ... end` est un exemple de structure de contrôle d'exécution utilisable dans un M-file. Les exemples d'utilisation sont suffisants pour comprendre

le fonctionnement de cette structure, qui est classique.

La structure de contrôle `for ... end` permet de faire des boucles. Là encore, son utilisation est similaire à celle qui en est faite dans le langage *C*. Sa syntaxe générale est la suivante :

```
for v = expression
    instructions
end
```

ou bien de façon plus compacte (mais moins lisible)

```
for v = expression, instructions, end
```

En général l'expression sera quelque-chose du type  $m:n$  ou  $m:p:n$ , comme dans l'exemple de la fonction `fibonacci` traité plus haut, mais si l'expression est une matrice, la variable `v` se verra assigner successivement les colonnes de cette matrice; l'exemple suivant illustre ce principe :

```
--> A=[1 2 3;4 5 6];
--> for v=A, x=v.^2, end
x =
     1
    16

x =
     4
    25

x =
     9
    36
```

La structure de contrôle `while ... end` permet de faire des boucles quand le nombre d'itérations n'est pas connu à priori. Voici par exemple une fonction faisant la division euclidienne de deux entiers :

```
function [quotient,reste]=divEucl(p,q)

reste=p;
quotient=0;
while reste>=q
    reste=reste-q;
    quotient=quotient+1;
```

```
end

endfunction
```

Tapez cette fonction dans l'éditeur, exécutez le script et expérimentez-le comme ceci

```
-->[q,r] = divEucl(7,2)
r =

    1.
q =

    3.
```

Notez que quand vous appelez une fonction, les variables utilisées en paramètres de sortie ont des noms différents de ceux utilisés à l'intérieur de la fonction.

## 4.4 Interaction avec l'utilisateur

Il est possible, dans un script ou une fonction, de faire entrer interactivement des données par l'utilisateur. Par exemple, dans la fonction calculant les premiers termes de la suite de *Fibonacci*, il serait intéressant de rentrer interactivement la valeur de  $n$ . Cela peut se faire à l'aide de la fonction `input`. Voici comment modifier le script `fibonacci.sce` : il suffit juste de remplacer la ligne

```
n=10;
```

par la ligne

```
n=input("Nombre de termes desire ? ");
```

Vous pouvez expérimenter le fonctionnement de cette fonction en exécutant le script à nouveau grâce au bouton 

On peut facilement générer un menu lorsque l'on a besoin que l'utilisateur fasse un choix parmi un certain nombre d'options, avec la commande `x_choose`, qui s'utilise de la façon suivante :

```
--> choice=x_choose(["French fries";
"Spaghetti Bolognese";"kaoya"],"Today's menu")
```

Le menu qui apparaît doit avoir l'aspect suivant : Le premier argument de `x_choose` est une matrice de chaînes de caractères contenant les textes associés aux diverses options. Le deuxième argument est le titre du menu apparaissant au dessus des boutons. Après un double-clic de souris sur l'une des options, la fonction renvoie le numéro de l'option choisie.



FIGURE 4.2 – Menu obtenu avec la commande `x_choose`



# Chapitre 5

## Les graphiques

Ce chapitre donne un bref aperçu des possibilités graphiques de Scilab, qui sont assez étendues : Scilab est capable de générer des graphiques à deux et à trois dimensions, d'agir facilement sur les couleurs et les types de lignes utilisés, entre autres.

### 5.1 Les graphiques à deux dimensions

#### 5.1.1 La commande `plot`

La commande utilisée pour générer un graphe en deux dimensions est la commande `plot`. Voici un premier exemple d'utilisation :

```
--> t=0:%pi/4:2*%pi;  
--> plot(t,sin(t))
```

La syntaxe est claire : `plot(x,y)` permet de tracer une courbe reliant les points dont les coordonnées sont données dans les vecteurs `x` pour les abscisses et `y` pour les ordonnées. Une première remarque s'impose : les points sont reliés par des segments de droite, et plus on prendra de points intermédiaires, plus le tracé sera fidèle, comme le montre l'exemple suivant (la commande `clf` permet d'effacer le contenu de la fenêtre graphique, car sinon par défaut tous les graphes se superposent) :

```
--> clf  
--> t=0:%pi/16:2*%pi;  
--> plot(t,sin(t))
```

Ici on a tracé le graphe d'une fonction, mais on peut aussi tracer une courbe paramétrique

$$\begin{aligned}x &= f(t), \\y &= g(t),\end{aligned}$$

pour  $t \in [a, b]$ .

Par exemple pour un cercle :

```
--> clf
--> t=0:%pi/32:2*%pi;
--> plot(cos(t),sin(t))
--> set(gca(),"isoview","on")
```

Notez la commande `set(gca(),"isoview","on")` qui permet d'imposer une échelle identique sur les deux axes de coordonnées.

Il est possible de superposer deux courbes sur le même graphique en un seul appel à la commande `plot` :

```
--> clf
--> plot(t,cos(t),t,sin(t))
```

Il est possible de changer les couleurs et les type de lignes utilisés par Scilab : par exemple si l'on veut que la courbe du sinus soit en rouge et que celle du cosinus soit en vert, il suffit de modifier la commande précédente de cette façon

```
--> clf
--> plot(t,cos(t),"g",t,sin(t),"r")
```

On peut aussi ne faire apparaître que les points et ne pas les relier par des segments de droite. Dans ce cas on peut repérer chaque point par un symbole (point, rond, étoile, croix). Par exemple

```
--> clf
--> plot(t,cos(t),"g^-",t,sin(t),"ro-")
```

Donc on peut modifier couleur et types de tracé (ligne ou symbole) en faisant suivre chaque couple  $x, y$  d'une chaîne de caractères entourée de guillemets composée de deux symboles précisant la couleur et le symbole.

Le tableau ci-dessous donne les symboles et les couleurs possibles :

Symbol	Color	Marker	Linetype
y	yellow	. dot	- plain line
m	magenta	o circle	- dashed line
c	cyan	x cross	-. dashdot line
r	red	+ plus	: dotted line
g	green	* star	
b	blue	d diamond	
w	white	^ triangle	
k	black	v triangle	

Il suffit de taper

```
--> help plot
```

pour en savoir plus.

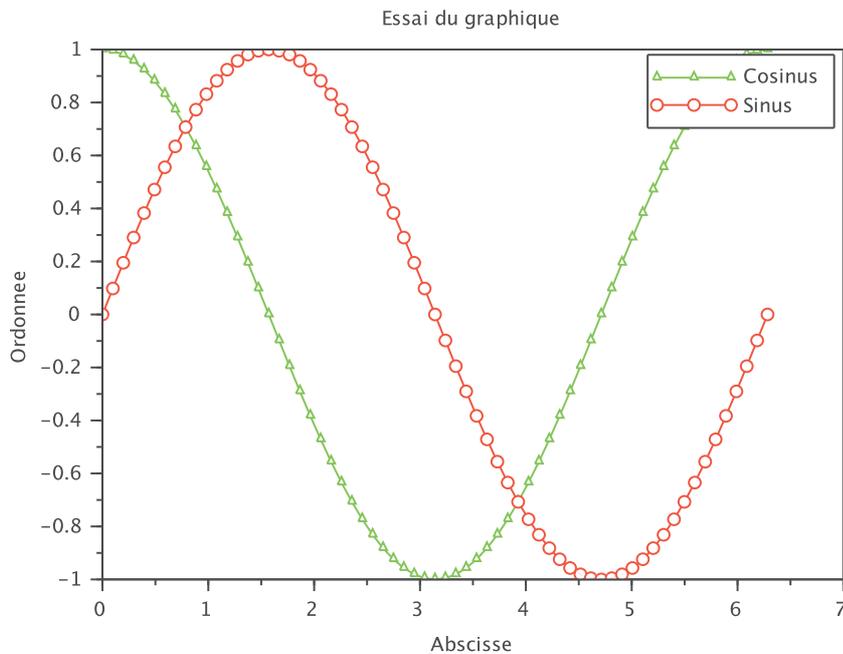


FIGURE 5.1 – Résultat de la commande plot

### 5.1.2 Titres et légendes

On peut facilement ajouter des légendes sur les axes et un titre à une figure :

```
--> xlabel("Abscisse")  
--> ylabel("Ordonnee")  
--> title("Essai du graphique")
```

On peut même ajouter une légende permettant de préciser à quoi correspond chaque courbe :

```
--> legend("Cosinus", "Sinus")
```

Il suffit de préciser les chaînes de caractères dans le même ordre que les couples abscisses-ordonnées dans la commande plot.

## 5.2 Manipuler plusieurs graphiques

On peut aisément disposer de plusieurs fenêtres graphiques. La fenêtre qui a été créée lors du premier appel à la fonction `plot`, et dans laquelle ont été effectuées toutes les sorties graphiques, est la fenêtre numéro 0. Si l'on désire créer une nouvelle fenêtre, de manière par exemple à avoir deux graphiques à l'écran, il suffit de taper (pour « set current figure »)

```
--> scf(1)
```

Cette commande crée une deuxième fenêtre dont le numéro est 1. Il est possible de créer ainsi autant de fenêtres que l'on désire. Quand il y a plusieurs fenêtres, il suffit d'activer la fenêtre ou l'on désire faire une sortie graphique avant de taper la commande qui produira cette sortie, par exemple :

```
--> t=linspace(-%pi,%pi,32);
--> scf(0); clf; plot(t,sin(t))
--> scf(1); clf; plot(t,cos(t))
```

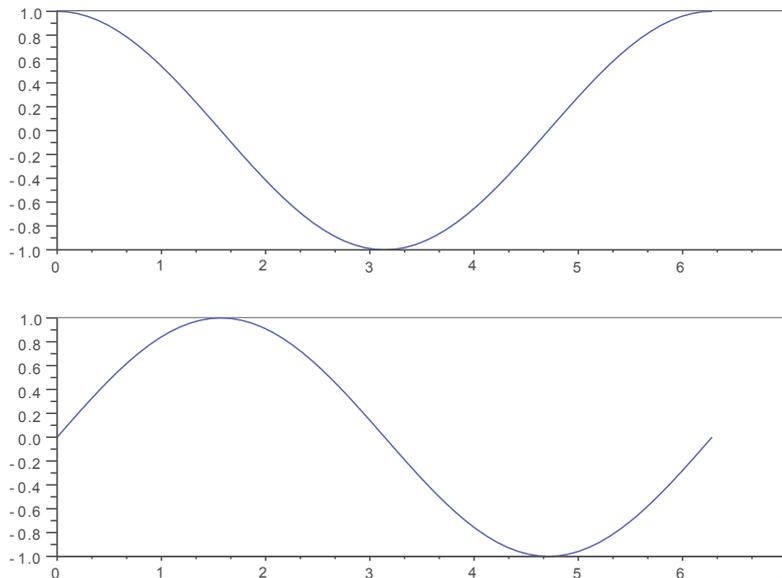


FIGURE 5.2 – Résultat obtenu avec la commande `subplot`

Une fenêtre peut aussi intégrer plusieurs graphes non superposés gr,ce à la commande `subplot`, qui s'utilise de la façon suivante :

```
--> clf
--> subplot(2,1,1)
--> plot(t,cos(t))
```

```
--> subplot(2,1,2)
--> plot(t,sin(t))
```

La commande `subplot(n,m,k)` permet de subdiviser la fenêtre en  $n*m$  zones,  $n$  portions verticalement et  $m$  portions horizontalement. Ces zones sont numérotées de gauche à droite et de haut en bas. La valeur de  $k$  permet de spécifier dans quelle zone on désire faire un graphique.

## 5.3 Les graphiques à trois dimensions

### 5.3.1 Les courbes

Il est par exemple possible de tracer des courbes dans l'espace avec la commande `param3d`, par exemple une hélice :

```
--> clf
--> t=0:%pi/32:8*pi;
--> param3d(cos(t),sin(t),t)
```

Notez au passage la commande `clf` permettant d'effacer la fenêtre graphique.

### 5.3.2 Les surfaces

Scilab permet de générer très facilement des graphiques à trois dimensions, par exemple, des graphes de fonctions de deux variables

$$z = f(x,y),$$

représentés dans l'espace, ou bien des graphes paramétriques

$$\begin{aligned}x &= f(t,s), \\y &= g(t,s), \\z &= h(t,s),\end{aligned}$$

pour  $(t,s) \in [a,b] \times [c,d]$ .

Voici un premier exemple permettant de tracer le graphe de la fonction

$$f(x,y) = \cos\left(\pi\sqrt{x^2+y^2}\right),$$

sur le domaine de variation  $(x,y) \in [-1,1] \times [-2,2]$ . Comme pour les courbes en 2D il faut choisir un pas d'échantillonnage, ici en fait deux puisque le domaine  $[-1,1] \times [-2,2]$  est bidimensionnel. On va par exemple prendre 10 points dans la direction  $x$  et 20 points dans la direction  $y$  :

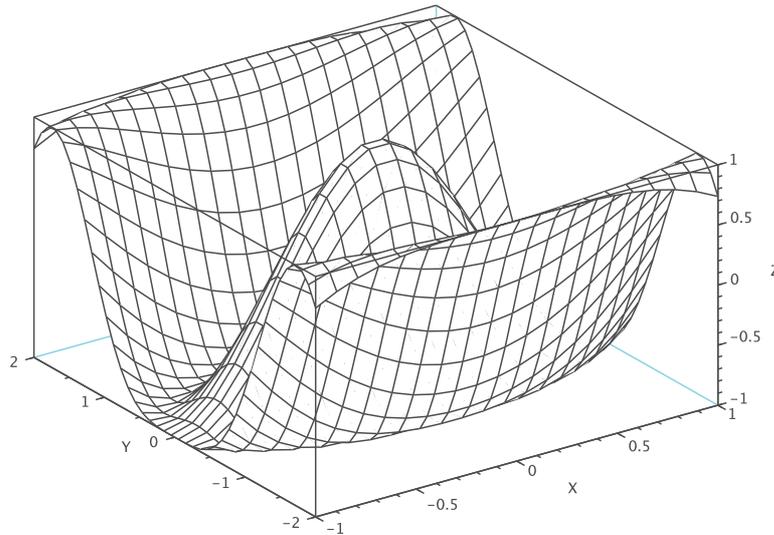


FIGURE 5.3 – Résultat de la commande mesh

```
--> x=linspace(-1,1,20);
--> y=linspace(-2,2,40);
```

Ces deux commandes nous ont permis d’obtenir deux vecteurs  $(x_i)_{i=1..20}$  et  $(y_j)_{j=1..40}$ , mais ceci est insuffisant car pour évaluer la fonction  $f(x,y)$  sur des points répartis dans le rectangle  $[-1, 1] \times [-2, 2]$  nous avons plutôt besoin des couples  $(x_i, y_j)$  pour  $(i, j) \in \{1..20\} \times \{1..40\}$ . En d’autres termes, on a besoin d’une “matrice de couples”  $(x_i, y_j)$  de taille  $20 \times 40$ . Puisque Scilab n’est pas capable de traiter un tel objet mathématique, il fournit plutôt deux matrices, une pour les  $x$  et l’autre pour les  $y$ . La commande permettant de générer ces deux matrices est la commande `meshgrid`. Pour notre exemple on l’utilise de la façon suivante :

```
--> [X,Y]=meshgrid(x,y);
```

On peut ensuite calculer en une seule instruction les valeurs de  $f$  aux points  $(x_i, y_j)$  et dessiner la surface obtenue avec la commande `mesh` :

```
--> clf
--> Z=cos(%pi*sqrt( X.^2 + Y.^2 ));
--> mesh(X,Y,Z)
```

La commande `mesh` dessine une espèce de “grillage” en reliant les points voisins.

Il est possible d’obtenir un “rendu” plus sympathique à l’aide des commandes suivantes :

```
--> clf
--> set(gcf(),"color_map",jetcolormap(128))
--> surf(X,Y,Z)
--> set(gcf(),"color_flag",3)
```

La commande `surf` permet de dessiner la surface, mais en donnant à chaque point de la surface une valeur (par défaut égale à la coordonnée  $z$ ) (voir sur la figure 5.3.2). La commande `set(gcf(),"color_map",jetcolormap(128))` permet de changer la table des couleurs, qui permet d'associer une couleur à la valeur associée à chaque point de la surface. La commande `set(gcf(),"color_flag",3)` permet d'interpoler les couleurs dans chaque triangle. Cela permet d'avoir un aspect "lissé", plus agréable à l'oeil, sans avoir à prendre plus de points pour générer la surface.

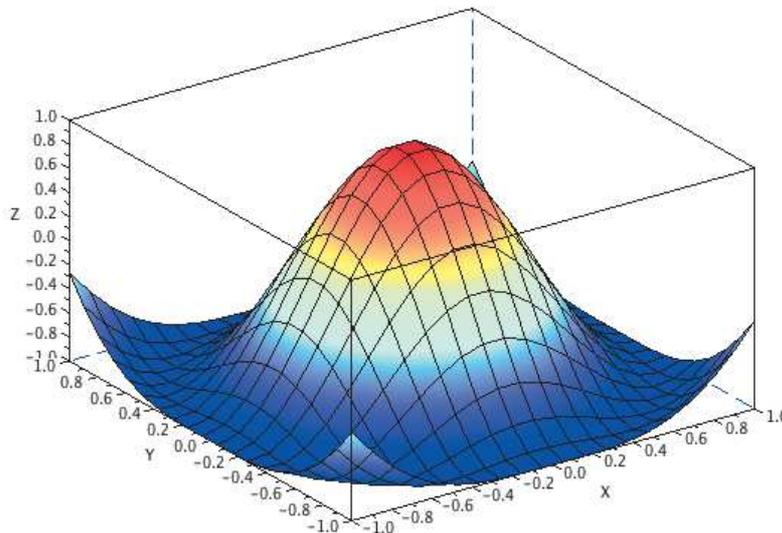


FIGURE 5.4 – Résultat de la commande `surf`

Vous pouvez sauver cette image dans différents formats de fichiers, en fonction de l'utilisation prévue. Par exemple, au format pdf

```
--> xs2pdf(0,"image.pdf")
```

le 0 étant ici le numéro de la fenêtre graphique.



# Chapitre 6

## Import et export de données

### 6.1 Lecture d'un fichier Excel (.xls)

Le type fondamental de données de Scilab est la matrice (ou tableau). Ce type est très proche des feuilles Excel destinées à créer des tables de données. De nombreux autres logiciels utilisent des formats d'échange voisins des concepts utilisés sur Excel. L'échange de données peut se faire de plusieurs manières :

- par la méthode "copier/coller" d'un logiciel vers l'autre (peu conseillé car le résultat est en général assez aléatoire)
- par enregistrement sur disque (fichier) depuis l'application source des données puis relecture sur l'autre application.

Pour lire un fichier Excel (classeur Excel 97-2004 .xls) nous allons utiliser la fonction `readxls`, mais au préalable nous allons télécharger le fichier :

```
--> getURL("http://www.utc.fr/~mottelet/scilab/data.xls")
```

Le fichier `data.xls` devrait normalement se trouver maintenant dans le répertoire courant de votre session Scilab. Pour le lire dans Scilab il suffit de taper

```
--> f=readxls("data.xls")
```

```
f =
```

```
Cell 1: 26x11
```

```
Cell 2: 23x5
```

```
Cell 3: 25x5
```

```
Cell 4: 87x5
```

```
Cell 5: 77x5
```

```
Cell 6: 82x5
```

```
Cell 7: 61x5
```

```
Cell 8: 55x5
```

```
Cell 9: 68x5
```

```

Cell 10: 116x5
Cell 11: 126x5
Cell 12: 117x5
Cell 13: 124x5
Cell 14: 118x5
Cell 15: 126x17

```

La variable `f` est de type structurée avec des champs permettant d'accéder aux feuilles individuelles, sous forme de matrices de chaînes de caractères ou de nombres, suivant l'usage qui doit en être fait. On peut par exemple visualiser le contenu de la quatrième feuille (on ne montre ici que les 4 premières lignes)

```

--> f(1)
ans =
      column 1 to 6

!Bubble n   Time (s)   Time (h)   Temp (C)   Press (kPa)   !
!0           2143       0.5952778  22.3       101.3         !
!           !           !           !           !           !
!1           10876       3.0211111  21.8       101.3         !
!           !           !           !           !           !
!2           32607       9.0575     21.2       101.3         !
.
.
.

```

et les valeurs numériques peuvent être récupérées ainsi

```

--> t4=f(4).value(2:$,3)
--> v4=f(4).value(2:$,1)*0.01
--> t5=f(5).value(2:$,3)
--> v5=f(5).value(2:$,1)*0.01

```

ce qui permet, par exemple, de représenter ici un volume cumulé de méthane en fonction du temps

```

--> plot(t4,v4,'-^',t5,v5,'-v')
--> legend(f(4).name,f(5).name,4)
--> xlabel("t (h)")
--> ylabel("vol (NmL)")

```

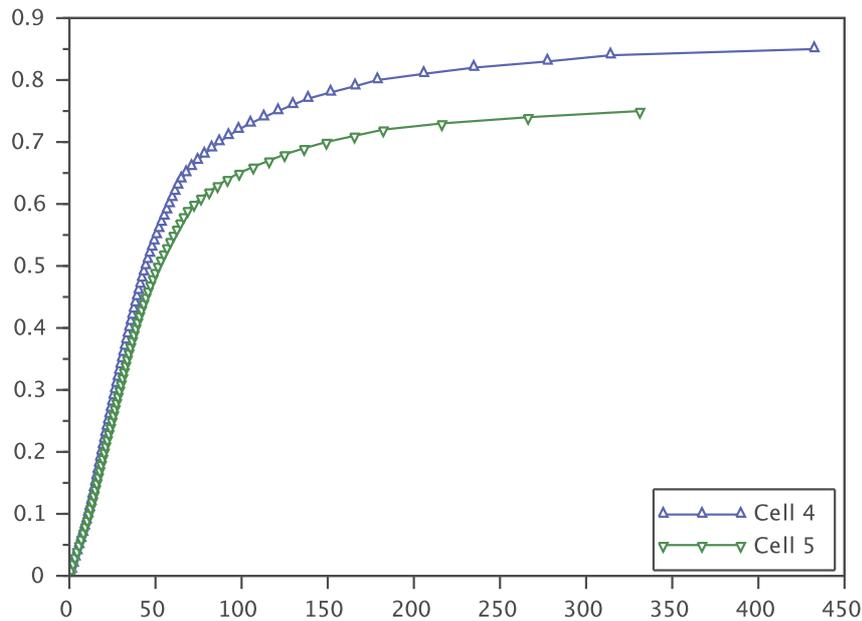


FIGURE 6.1 – Graphique réalisé avec des données extraites d'un fichier Excel

## 6.2 Lecture et écriture d'un fichier .csv

Le format CSV (Comma Separated Values) est un format texte (par opposition à format binaire) qui permet d'échanger facilement des données avec un tableur. Il faut cependant faire attention aux paramètres linguistiques par défaut, en particulier pour la décimale (virgule ou point). On peut par exemple sauvegarder ainsi une matrice aléatoire destinée à être relue dans un tableur dont la langue est réglée en français :

```
--> a=rand(5,5)
--> csvWrite(a,"matrice1.csv",";",",",",")
```

Ici on impose que le séparateur de colonne soit un point-virgule et que la décimale soit une virgule. Par défaut, comme l'indique le nom du format CSV, le séparateur est une virgule et la décimale un point (convention anglo-saxonne). Par exemple ici on écrit puis on relit un fichier respectant cette convention

```
--> csvWrite(a,"matrice2.csv")
--> csvRead("matrice2.csv")
```

Puisqu'ils sont au format texte, les deux fichiers `matrice1.csv` et `matrice2.csv` peuvent être ouverts dans l'éditeur de Scilab pour les comparer.