Introduction à SCILAB

S. MOTTELET

ÉQUIPE DE MATHÉMATIQUES APPLIQUÉES
DÉPARTEMENT DE GÉNIE INFORMATIQUE
UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

Contents

Chapter 1

Pour bien démarrer

Ce premier chapitre explique les quelques rudiments à connaître pour commencer à utiliser SCILAB. Il explique

- Comment entrer une matrice, la syntaxe des commandes SCILAB et les variables
- Comment avoir des informations sur l'espace de travail, comment sauver tout ou partie de l'espace de travail et comment quitter SCILAB.
- Les nombres, les expressions arithmétiques et les fonctions usuelles
- La façon dont SCILAB affiche les résultats

Vous pouvez dès maintenant lancer une session SCILAB sur votre machine pour traiter les exemples qui vous seront proposés. Vous devriez avoir à l'écran quelque-chose comme ça :

```
           ===========
           S c i l a b
           ===========


                scilab-2.5.1
          Copyright (C) 1989-2000 INRIA


Startup execution:
  loading initial environment
  loading plotlib v0.15

-->
```

Le signe `-->` signifie que SCILAB est en attente d'une commande. Dans le texte, chaque exemple est repérable facilement par les caractères utilisés (caractères de `machine à écrire`), et chaque commande à entrer au clavier sera toujours précédée du signe `-->`.

1.1 Entrer une matrice

SCILAB travaille essentiellement avec un seul type d'objet, les matrices rectangulaires contenant des nombres réels ou complexes et eventuellement des caractères. Dans certaines situations, une signification particulière est donnée aux matrices 1×1, qui sont des *scalaires*, et aux matrices avec une seule ligne ou une seule colonne, qui sont des *vecteurs*. Les opérations et commandes de SCILAB agissent donc sur ce type d'objet.

Il y a plusieurs façons d'entrer une matrice dans l'espace de travail de SCILAB :

- Entrer au clavier une liste explicite d'éléments.
- Générer cette matrice par une suite de commandes SCILAB.
- Charger cette matrice à partir d'un fichier de données externe.

Le langage SCILAB ne contient pas de déclaration de dimension ou de type (à l'opposé de langages comme le *Pascal*). La place mémoire nécessaire à une matrice est automatiquement allouée au moment de sa création.

La façon la plus simple d'entrer une petite matrice est d'entrer au clavier la liste de ses éléments, en adoptant les conventions suivantes :

- Les éléments d'une même ligne sont séparés par des espaces ou des virgules
- La liste des éléments doit être entourée de crochets, [ ]
- Chaque ligne (sauf la dernière) se termine par un ; (point-virgule)

Par exemple, la commande suivante

```
--> A = [1  2  3;  4  5  6;  7  8  9]
```

produit la sortie suivante

```
A  =

!   1.   2.   3. !
!   4.   5.   6. !
!   7.   8.   9. !
```

La matrice A est gardée en mémoire pour un usage ultérieur.

Une grande matrice peut être entrée en séparant chaque ligne par un retour-chariot pour remplacer le point-virgule. Par exemple la matrice précédente peut être entrée comme ceci :

```
--> A = [1  2  3
    4  5  6
    7  8  9]
```

1.2 Les éléments d'une matrice

Les élément d'une matrice peuvent être n'importe quelle expression SCILAB; par exemple

```
--> x = [-1.3  sqrt(3)  (1+2+3)*4/5]
```

produit la sortie

```
x  =

! - 1.3   1.7320508   4.8 !
```

Les éléments individuels d'une matrice peuvent être référencés avec leur indice, précisé entre deux parenthèses, (). Si nous continuons notre exemple

```
--> x(5) = abs(x(1))
```

donne la sortie

```
x  =

! - 1.3   1.7320508   4.8   0.   1.3 !
```

On notera que la taille de `x` a été augmentée automatiquement pour pouvoir ajouter le nouvel élément, et que les éléments indéfinis ont par défaut été initialisés à zéro.

1.3 Les commandes SCILAB et les variables

SCILAB est un langage qui possède une syntaxe simple mais bien définie. Une commande sera toujours de la forme

```
variable = expression
```

ou plus simplement

```
expression
```

Une expression peut être composée de matrices séparées par des opérateurs, de fonctions et de noms de variables. L'évaluation de l'expression produit une matrice qui est affichée à l'écran et qui est affectée à `variable`. Si le nom de la variable ainsi que le signe = sont omis, SCILAB crée automatiquement une variable portant le nom `ans` (pour answer). Par exemple, l'expression suivante

```
--> 1900/81
```

produit l'affichage suivant

```
ans =

   23.4568
```

Une commande se termine par l'appui sur la touche Return (ou Enter). Cependant, si le dernier caractère d'une ligne de commande est ; (un point-virgule), cela supprime l'affichage du résultat, sans toutefois affecter l'exécution de la commande. On utilisera cette option par exemple lors de calculs intermédiaires dont le résultat n'a pas besoin d'être affiché. Par exemple

```
--> y = sqrt(3)/2;
```

affecte à `y` la valeur de $\sqrt{3}/2$ mais n'affiche pas le résultat.

Un nom de variable doit obligatoirement commencer par une lettre, suivie par un nombre quelconque de chiffres et de lettres. Attention : SCILAB ne prend en compte que les 19 premiers caractères[1] du nom d'une variable. SCILAB fait aussi la différence entre les majuscules et les minuscules. Par exemple les deux commandes suivantes

```
--> z = log(2);
--> Z
```

produisent l'affichage suivant

```
!--error    4
  undefined variable : Z
```

1.4 Obtenir des informations sur l'espace de travail

Les exemples proposés précédemment ont créé des variables qui sont stockées dans *l'espace de travail* de SCILAB. Pour obtenir la liste de ces variables, il suffit d'entrer la commande

```
--> who
your variables are...

y        x        z       A       MSDOS    home     PWD
percentlib fraclablib soundlib  xdesslib  utillib   tdcslib   siglib
s2flib    roblib    optlib   metalib   elemlib   commlib   polylib
autolib   armalib   alglib   mtlblib   SCI      %F      %T
%z       %s       %nan     %inf     old      newstacksize
%t       %f       %eps     %io      %i      %e      %pi
using      3979 elements  out of   1000000.
     and       43 variables out of      1023
```

[1] Cela est en général suffisant !

On retrouve au début de cette liste les 4 variables qui ont été crées dans les exemples précédents. Pour connaître la taille de ces variables, il suffit d'utiliser la commande

```
   --> whos()
Name                Type            Size            Bytes

whos                compiled function                4984
y                   constant matrix    1 by 1        24
x                   constant matrix    1 by 5        56
z                   constant matrix    1 by 1        24
A                   constant matrix    3 by 3        88
.
.
.
```

(on ne reproduit ici que le début de la liste qui apparaît à l'écran). Chaque élément d'une matrice réelle occupe 8 octets (bytes) de la mémoire, et pour chaque matrice 16 octets supplémentaires sont utilisés pour stocker diverses informations (taille, etc.). Ceci explique par exemple que la matrice A, qui est de taille 3×3, occupe $9 \times 8 + 16 = 88$ octets.

1.5 Quitter et sauver l'espace de travail

Pour quitter SCILAB il suffit de taper `quit` ou `exit`. Lorsqu'une session SCILAB est terminée, toutes les variables présentes dans l'espace de travail sont perdues. Il est possible de sauvegarder l'espace de travail avant de quitter en tapant

```
--> save work.dat
```

Cette commande sauve toutes les variables dans votre répertoire dans un fichier appelé `work.dat`. Il est possible lors d'une session ultérieure de récupérer le contenu de ce fichier en tapant la commande `load work.dat`.

Il est possible d'utiliser les commandes `save` et `load` avec d'autres noms de fichier, ou bien de sauvegarder uniquement certaines variables. La commande

```
--> save('work.dat',A,x)
```

ne sauve dans ce même fichier que les variables A et x. La commande `load work.dat` permettra de récupérer toutes les variables sauvegardées dans ce fichier (ici A et x).

1.6 Nombres, expressions arithmétiques et fonctions usuelles

SCILAB utilise la notation décimale classique, avec un point décimal optionnel (si le nombre est entier), précédée du signe moins pour les nombres négatifs. Il est possible d'inclure une puissance de dix en facteur. Des exemples de nombres légaux sont

```
 3          -99          0.0001
 9.37821312   1.6023E-20   6.02252e23
```

La précision relative des nombres est donnée par la variable permanente `eps` (permanente veut dire que vous ne pouvez pas la faire disparaître de l'espace de travail)

```
--> %eps

eps =

  2.220e-16
```

Cette valeur signifie que la précision est à peu de choses près de 16 chiffres significatifs. On peut illustrer ceci par l'exemple suivant

```
--> (1 + 1e-16) - 1

ans =

   0.
```

Il faut donc toujours avoir à l'esprit que l'arithmétique sur ordinateur n'est jamais exacte. L'étendue des nombres représentables va de 10^{-308} à 10^{308}.

On peut construire des expressions avec les opérateurs usuels, qui respectent l'ordre de priorité classique

- `^` élévation à une puissance
- `/` division à droite
- `\` division à gauche
- `*` multiplication
- `+` addition
- `-` soustraction

Nous verrons dans un prochain paragraphe comment ces opérations sont utilisables sur les matrices. SCILAB dispose en standard des fonctions mathématiques usuelles que l'on trouve sur toute calculatrice moderne, nous ne citons ici que les plus classiques

`sqrt`	racine carrée
`log`	logarithme
`log10`	logarithme décimal
`sin`	sinus
`cos`	cosinus
`tan`	tangente
`atan`	arc tangente
`exp`	exponentielle
`cosh`	cosinus hyperbolique
`floor`	partie entière
`round`	arrondi à l'entier le plus proche
`abs`	valeur absolue ou module pour les complexes
`real`	partie réelle d'un complexe
`imag`	partie imaginaire d'un complexe
`modulo`	reste d'une division euclidienne

Ces fonctions agissent bien sûr sur les nombres complexes. Par exemple

```
--> sqrt(-1)

   ans =

      i

--> exp(%i*%pi/3)

ans  =

0.5 + 0.8660254i
```

Certaines fonctions renvoient simplement des valeurs spéciales. Par exemple la variable permanente `%pi`

```
--> %pi

ans =

  3.1416
```

renvoie la valeur de π, pré-calculée comme étant la valeur de `4*atan(1)`. La fonction `Inf`, pour désigner $+\infty$, est disponible dans un petit nombre de calculateurs ou de systèmes de calcul. La valeur renvoyée par cette fonction peut générée par certaines opérations, par exemple une division par zéro, ou un déplacement de capacité (overflow)

```
--> 1/0

Warning :division by zero...

ans  =

   Inf
```

On notera au passage qu'une division par zéro ne "plante" pas le programme mais donne juste un "warning". On peut utiliser `%inf` dans les calculs :

```
--> 1/%inf

ans =

  0
```

Une variable permanente similaire à `Inf` est la variable `%nan`. Il s'agit de la valeur renvoyée lors d'une opération illicite, par exemple

```
--> 0/0

Warning :division by zero...

ans  =

   Nan

--> %inf/%inf

ans  =

   Nan
```

En général, lorsque ces valeurs sont générées par un programme, ce n’est pas bon signe et cela doit attirer l’attention sur un éventuel “bug” (erreur de programmation).

Chapter 2

Les opérations sur les matrices

2.1 La transposition

Le caractère spécial ' (prime ou apostrophe) désigne l'opération de transposition. Les commandes suivantes

```
--> A = [1  2  3;  4  5  6;  7  8  0]
--> B = A'
```

donnent les résultats

```
A =

    1    2    3
    4    5    6
    7    8    0

B =

    1    4    7
    2    5    8
    3    6    0
```

et la commande

```
--> x = [-1  0  2]'

x =

  !  -1. !
  !   0. !
  !   2. !
```

L'opération de transposition transpose les matrices au sens classique. Si Z est une matrice complexe, alors Z' désigne la transposée conjuguée de Z.

2.2 Addition et soustraction

Les opérateurs + et - agissent sur les matrices. Ces opérations sont valides dès que les dimensions des matrices sont les mêmes. Par exemple avec les matrices de l'exemple précédent l'addition

```
--> A + x
      !--error    8
inconsistent addition
```

n'est pas valide car A est 3×3 et x est 3×1. Par contre l'opération suivante est valide :

```
--> C = A + B

C =

  !   2.    6.   10. !
  !   6.   10.   14. !
  !  10.   14.    0. !
```

L'addition et la soustraction sont aussi définies si l'un des opérandes est un scalaire, c'est à dire une matrice 1×1. Dans ce cas le scalaire est additionné ou soustrait de tous les éléments de l'autre opérande, par exemple :

```
--> z = x - 1

z =

   ! -2. !
   ! -1. !
   !  1. !
```

2.3 Multiplication

Le symbole * désigne l'opérateur de multiplication des matrices. Cette opération est valide dès que les dimensions des opérandes sont compatibles, à savoir, le nombre de colonnes de l'opérande de gauche doit être égal au nombre de lignes de l'opérande de droite. Par exemple l'opération suivante n'est pas valide

```
--> x*z
    !--error   10
inconsistent multiplication
```

Par contre la commande suivante

```
--> x' * z

ans =

    4.
```

donne le produit scalaire de `x` et `z`. Une autre commande valide est la suivante :

```
--> b = A*x

b =

    !  5. !
    !  8. !
    ! -7. !
```

La multiplication d'une matrice par un scalaire est bien sûr toujours valide :

```
 --> A*2

ans =

   !   2.    4.    6. !
   !   8.   10.   12. !
   !  14.   16.    0. !
```

2.4 Inversion d'une matrice et division

On obtient facilement l'inverse d'une matrice carrée avec la commande `inv` :

```
--> B = inv(A)

 B  =

! - 1.7777778   0.8888889  - 0.1111111 !
!   1.5555556  - 0.7777778   0.2222222 !
! - 0.1111111   0.2222222  - 0.1111111 !

--> C=B*A

 C  =

!   1.   0.   0. !
!   0.   1.   0. !
```

```
!   0.   0.   1. !

--> C(2,1)

 ans  =

- 1.110E-16
```

On notera au passage les erreurs dues à la précision finie des calculs.

La division de matrice est implantée dans SCILAB, et porte la signification suivante : l'expression `A/B` donne le résultat de l'opération AB^{-1}. Elle est donc équivalente à la commande `A*inv(B)`. Pour la division à gauche, l'expression `A\B` donne le résultat de l'opération $A^{-1}B$. Il faut donc respecter la compatibilité des dimensions des deux matrices pour que cette division ait un sens.

La division à gauche est utilisée classiquement lorsque l'on a besoin de résoudre un système linéaire. Par exemple si l'on désire résoudre le système linéaire $Ay = x$, on écrira

```
--> y = A\x

y  =

!   1.5555556 !
! - 1.1111111 !
! - 0.1111111 !
```

Lorsque SCILAB interprète cette expression, il n'inverse pas la matrice `A` avant de la multiplier à droite par `x`; il résout effectivement le système. On peut vérifier l'exactitude des résultats de la façon suivante :

```
--> A*y - x

ans =

  1.0E-15 *

!   0.2220446 !
!   0.2775558 !
!   0.        !
```

On notera comme pour l'exemple précédent la présence d'erreurs de l'ordre de la précision machine.

2.5 Opérations élément par élément

Les opérations usuelles sur les matrices peuvent être effectuées élément par élément; cela revient à considérer les matrices comme des tableaux de chiffres et non plus comme des objets mathématiques de l'algèbre linéaire. Pour l'addition et la soustraction les deux points de vues sont les mêmes puisque ces deux opérations agissent déjà élément par élément pour les matrices.

Le symbole `.*` désigne la multiplication élément par élément. Si `A` et `B` ont les mêmes dimensions, alors `A.*B` désigne le tableau dont les éléments sont simplement les produits des éléments individuels de `A` et `B`. Par exemple si on définit `x` et `y` de la façon suivante :

```
--> x = [1  2  3];  y = [4  5  6];
```

alors la commande

```
--> z = x .* y
```

produit le résultat

```
z =

    !  4.   10.   18. !
```

La division fonctionne de la même manière :

```
--> z = x ./ y

z =

  !   0.25   0.4   0.5 !
```

Le symbole `.^` désigne l'élévation à la puissance élément par élément :

```
--> x .^ y

ans =

   !  1.   32.   729. !

--> x .^ 2

ans =

    !  1.    4.    9. !
```

```
--> 2 .^ x

ans =

    !  2.    4.    8. !
```

On notera que pour `.^` l'un des deux opérandes peut être un scalaire.

2.6 Opérateurs relationnels

Six opérateurs relationnels sont disponibles pour comparer deux matrices de dimensions égales :

`<`	plus petit que
`<=`	plus petit ou égal
`>`	plus grand
`>=`	plus grand ou égal
`==`	égal
`~=`	différent

SCILAB compare les paires d'éléments correspondants; le résultat est une matrice de constantes booléennes, le `F` (false) représentant la valeur "faux" et le `T` (true) la valeur "vrai". Par exemple

```
--> 2 + 2 ~= 4

ans =

    F
```

Les opérateurs logiques permettent de voir dans une matrice la disposition des éléments vérifiant certaines conditions. Par exemple prenons la matrice

```
--> A = [1  -1  2; -2  -4  1; 8  1  -1]

A =

   !   1.   -1.    2. !
   !  -2.   -4.    1. !
   !   8.    1.   -1. !
```

La commande

```
--> P = (A < 0)
```

```
 P  =

! F T F !
! T T F !
! F F T !
```

renvoie une matrice P indiquant par des T les éléments négatifs de A.

2.7 Opérateurs logiques

Les opérateurs &, | et ~ désignent les opérateurs logiques “et”, “ou” et “non”. Il sont utilisés pour élaborer des expressions logiques. Par exemple si l’on prend la matrice de l’exemple précédent, la commande

```
--> P = (A < 0) & (modulo(A,2) == 0)

P =

! F F F !
! T T F !
! F F F !
```

permet de repérer dans A les éléments négatifs et multiples de 2.

2.8 Fonctions usuelles appliquées à une matrice

Les fonctions usuelles s’appliquant sur réels et complexes s’appliquent aussi élément par élément sur les matrices. Par exemple :

```
--> A = [0  1/4;  1/2  3/4]

A =

!   0.    0.25 !
!   0.5   0.75 !

--> cos(%pi*A)

ans =

!   1.   0.7071068 !
!   0.  - 0.7071068 !
```

Chapter 3

Manipulations des matrices et des vecteurs

3.1 Comment générer des matrices et des vecteurs

On peut facilement générer une matrice nulle de taille quelconque avec la commande `zeros` qui s'utilise comme ceci :

```
--> A = zeros(3,2)

A =

!   0.   0. !
!   0.   0. !
!   0.   0. !
```

Cette commande peut par exemple servir pour créer et initialiser une matrice.

On peut générer la matrice identité avec la commande `eye` de la façon suivante :

```
--> I = eye(3,3)

I =

!   1.   0.   0. !
!   0.   1.   0. !
!   0.   0.   1. !
```

On peut aussi générer des vecteurs particuliers à l'aide du caractère `:` (deux points), comme le montre l'exemple suivant :

```
--> x = 1:5
```

```
x =

    !  1.    2.    3.    4.    5. !
```

Cette commande a permis d'affecter au vecteur `x` les entiers de `1` à `5`. On peut aussi spécifier un incrément particulier autre que `1` de la façon suivante :

```
--> y = 0 : %pi/4 : %pi

y =

   !   0.   0.7853982   1.5707963   2.3561945   3.1415927 !
```

On notera que cette commande produit toujours un vecteur ligne. Il est bien sûr possible d'utiliser un incrément négatif :

```
--> y = 6:-1:1

y =

!   6.   5.   4.   3.   2.   1. !
```

On peut aisément créer des tables en utilisant cette commande (on verra au chapitre suivant que cela permet de créer rapidement des représentations graphiques). Par exemple :

```
--> x = (0:0.2:3)';
--> y=exp(-x).*sin(x);
--> [x y]

ans =

!   0.    0.      !
!   0.2   0.1626567 !
!   0.4   0.2610349 !
!   0.6   0.3098824 !
!   0.8   0.3223289 !
!   1.    0.3095599 !
!   1.2   0.2807248 !
!   1.4   0.2430089 !
!   1.6   0.2018104 !
!   1.8   0.1609759 !
!   2.    0.1230600 !
!   2.2   0.0895840 !
```

```
!   2.4   0.0612766 !
!   2.6   0.0382881 !
!   2.8   0.0203707 !
!   3.    0.0070260 !
```

Il existe une commande permettant de spécifier uniquement les valeurs minimum et maximum et le nombre de valeurs désirés :

```
--> k = linspace(-%pi,%pi,5)

k =

! - 3.1415927  - 1.5707963   0.   1.5707963   3.1415927 !
```

3.2 Comment manipuler les éléments d'une matrice

On peut faire référence aux éléments individuels d'une matrice en précisant leurs numéros de ligne et de colonne dans des parenthèses suivant le nom de la matrice. Prenons la matrice

```
--> A = [1  2  3;  4  5  6;  7  8  9]
```

La commande suivante permet de remplacer a_{33} par $a_{13} + a_{31}$:

```
--> A(3,3) = A(1,3) + A(3,1)

A =

!   1.   2.   3.  !
!   4.   5.   6.  !
!   7.   8.   10. !
```

On peut aussi facilement extraire une ligne ou une colonne d'une matrice; par exemple, la commande

```
--> v = A(:,1)

v =

   !  1. !
   !  4. !
   !  7. !
```

permet de mettre dans le vecteur `v` la première colonne de la matrice `A`. De la même manière, la commande

```
--> v = A(1,:)
```

permettra d'extraire la première ligne de `A`.

Un indice de ligne ou de colonne peut être un vecteur. Par exemple si nous prenons le vecteur

```
--> x = 0:2:8;
```

la commande suivante permet d'effectuer une permutation des éléments de `x` :

```
--> v = [3 5 1 2 4];
--> x(v)

ans =

    ! 4.    8.    0.    2.    6. !
```

Il est possible de construire une matrice à partir de matrices plus petites. Par exemple, pour ajouter une nouvelle ligne à la matrice `A` :

```
 --> l = [10  11  12];
 --> A = [A;  l]
```

Cela donne

```
   A =

!   1.    2.    3.  !
!   4.    5.    6.  !
!   7.    8.    10. !
!   10.   11.   12. !
```

On peut aussi facilement manipuler les sous-matrices d'une matrice, on l'a déjà vu plus haut pour le cas d'une ligne ou d'une colonne :

```
--> A(1:2,1:2) = eye(2,2)

A =

!   1.    0.    3.  !
!   0.    1.    6.  !
!   7.    8.    10. !
!   10.   11.   12. !
```

Ici on a remplacé la sous-matrice principale d'ordre 2 de `A` par la matrice identité.
On peut extraire une sous-matrice d'une matrice plus grande. Par exemple

```
 --> B = A( 1:2, : );
```

sélectionne les deux premières lignes et toutes les colonnes de `A` et les stocke dans une matrice `B`.
Pour en finir avec les matrices, on peut obtenir automatiquement leur taille (nombre de lignes et de colonnes) avec la fonction `size` :

```
--> size(A)

ans =

    !  4.    3. !

--> [n,m]=size(v)

m =

    5.

n =

    1.
```

La première de ces deux commandes renvoie un vecteur à deux composantes : la première est le nombre de lignes et la deuxième est le nombre de colonnes. La deuxième commande permet d'assigner ces deux valeurs à deux variables différentes, `n` pour les lignes et `m` pour les colonnes.

Chapter 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 crées 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 crées 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 :

$$\left\{ \begin{array}{rcl} u_0 & = & 1, \\ u_1 & = & 1, \\ u_{k+2} & = & u_{k+1} + u_k,\ k \geq 0 \end{array} \right.$$

Dans un premier temps, il faut créer le fichier qui va contenir le script. Pour cela, il faut selectionner la commande `Create File` (voir figure 4.1) dans le menu `File` du File

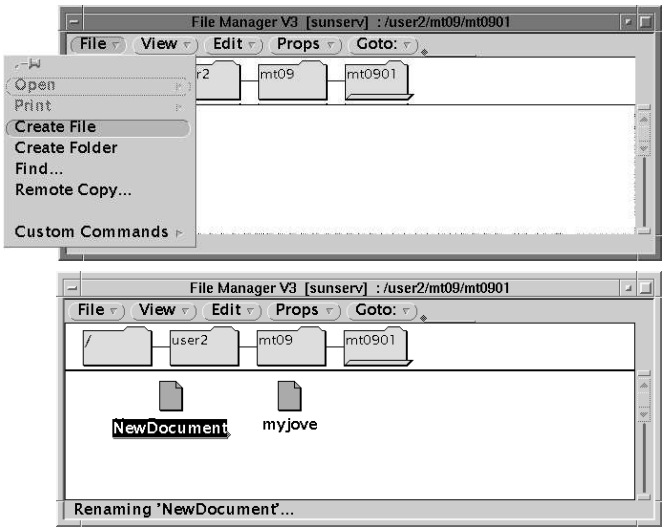

Figure 4.1: Création d'un fichier

Manager. Le fichier créé a par défaut le nom `NewDocument`. Pour le changer il suffit de taper le nouveau nom et d'appuyer sur `Return`. Ici nous choisirons le nom `fibo.sce`. Après avoir saisi le nom, il suffit de double-cliquer sur l'icône du fichier pour invoquer l'éditeur de texte; la fenêtre créée doit ressembler à celle qui se trouve sur la figure 4.2. Vous pouvez maintenant saisir les lignes suivantes dans la fenêtre de l'éditeur :

Figure 4.2: Fenêtre de l'éditeur de texte

```
// 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

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 menu `File` de l'éditeur).

Pour exécuter le script `fibo.sce`, il suffit de taper :

```
--> exec fibo.sce

u =

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

4.2 Les fonctions

Un fichier contenant le mot `function` 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 icone 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
```

```
// syntaxe : variable = fact(n)

if (n - floor(n) ~= 0) | n < 0
   error('erreur dans fact : l''argument doit etre 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 *Pascal*.
- 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 taper :

```
--> getf fact.sci
```

Une remarque importante : il est possible de définir plus d'une fonction dans un seul fichier. Dans ce cas, l'instruction `getf` charge toutes les fonctions définies dans le fichier. 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 *Pascal*. 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 `fibo` 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
```

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 `fibo.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 tapant à nouveau :

```
--> exec fibo.sce
```

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

```
--> choix = x_choose(['Saucisse Frites';'Boeuf Carottes';...
    'Couscous'],'Ce midi')
```

Le menu qui apparaît doit avoir l'aspect suivant :

Figure 4.3: Menu obtenu avec la commande `x_choose`

Le premier argument de `x_choose` est une matrice de chaines de caractères contenant les textes associés aux diverses options. Le deuxième argument est le titre du menu apparaissant au dessus des boutons. Après un clic de souris sur l'une des options, la fonction renvoie le numéro de l'option choisie.

Chapter 5

Les graphiques

Ce chapitre donne un bref appercu des possibilités graphiques[1] 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, etc.

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 :

```
--> 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 du type

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

[1] Une partie des possibilités graphiques présentées font appel à la librairie développée par Stéphane Mottelet, disponible à l'URL `http://www.dma.utc.fr/mottelet/myplot.html`

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

Par exemple pour une courbe de *Lissajous* :

```
--> t = 0:%pi/32:2*%pi;
--> plot(cos(t),sin(2*t))
```

Il est possible de superposer deux courbes sur le même graphique :

```
--> plot(t,cos(t),t,sin(t),'axis','equal')
```

Au passage, la paire `'axis','equal'` permet d'avoir un graphique avec la même échelle pour les deux axes.

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

```
--> 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

```
--> 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 d'apostrophes (le même caractère que pour la transposition des matrices) composée de deux symboles précisant la couleur et le symbole.
Le tableau ci-dessous donne les symboles et les couleurs possibles :

Symbole	**Couleur**	**Symbole**	**Type de ligne**
y	jaune	.	point
m	magenta	o	rond
c	cyan	x	croix
r	rouge	+	plus
g	vert	*	étoile
b	bleu	-	trait plein
w	blanc	d	diamant
k	noir	^	triangle
		v	triangle

Il suffit de taper

```
--> help plot
```

pour en savoir plus.

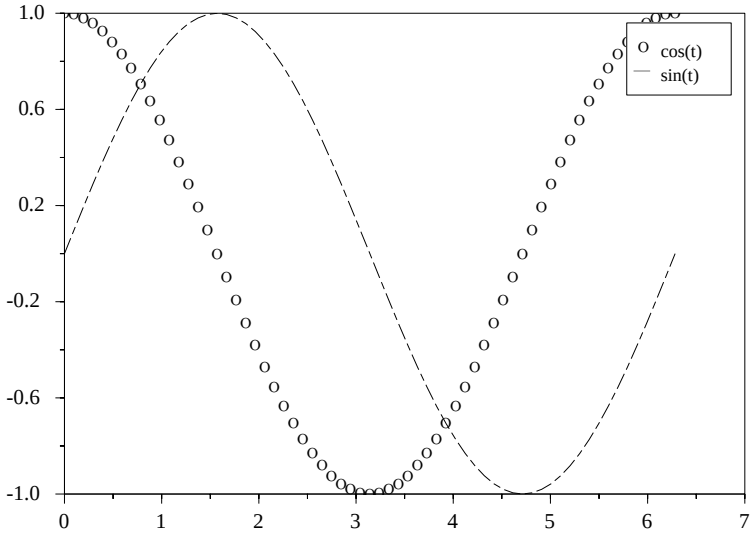

Figure 5.1: Résultat de la commande `plot`

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-ordonées dans la commande `plot`.

5.2 Les graphiques à trois dimensions

5.2.1 Les courbes

Il est par exemple possible de tracer des courbes dans l'espace avec la commande `plot3`, par exemple une hélice :

```
--> clf()
--> t = 0:%pi/32:8*%pi;
--> plot3(cos(t),sin(t),t)
```

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

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

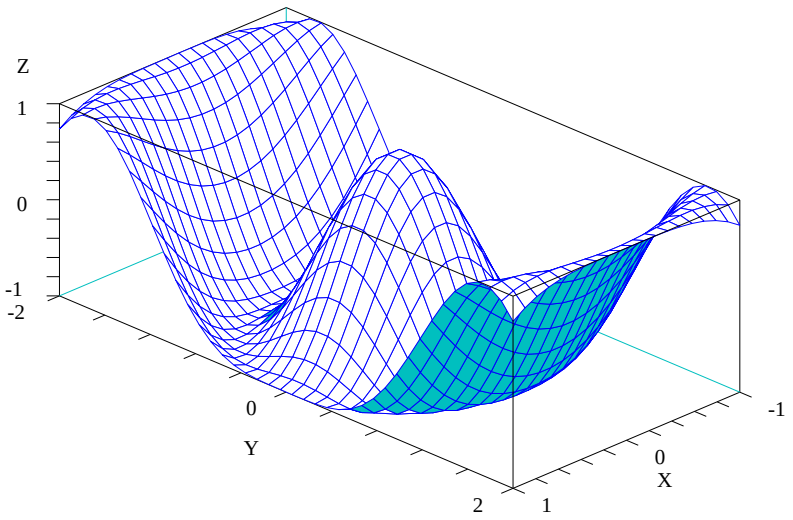

Figure 5.2: Résultat de la commande `mesh`

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 20 points dans la direction x et 40 points dans la direction y :

```
--> 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 $(y_j)_{j=1..40}$. Maintenant, définissons «en ligne» la fonction à tracer :

```
--> deff('z=maSurface(x,y)','z=cos(%pi*sqrt(x.^2+y.^2))');
```

On peut ensuite tracer la surface obtenue avec la commande `mesh` :

```
--> mesh(x,y,maSurface)
```

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

Pour imprimer ce graphique, il suffit de selectionner l'option `print` dans le menu `file` de la fenêtre graphique.

Il est possible d'avoir un autre type de visualisation de la surface : ici, la commande `surf` donne à chaque polygone une couleur proportionnelle à la valeur moyenne de z.

```
--> surf(x,y,maSurface)
```

Il est possible d'obtenir un "rendu" plus agréable à l'aide dela commande suivante :

```
--> surf(x,y,maSurface,'shading','flat')
```

La paire `'shading','flat'` permet de suprimer le contour des polygones.

5.3 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 a l'écran, il suffit de taper

```
--> fig(1)
```

Cette commande crée une deuxième fenêtre dont le numéro est `2`. 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 :

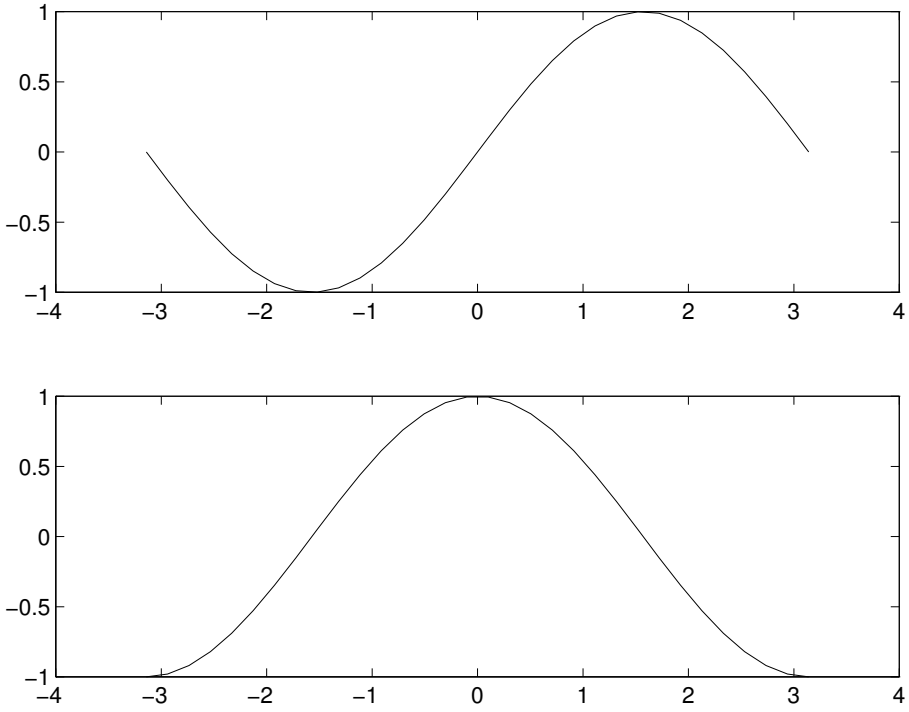

Figure 5.3: Résultat obtenu avec la commande `subplot`

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

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

```
--> 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.