Introduction à Scilab



www.scilab.org
scilab.discourse.group

Stéphane Mottelet

Laboratoire TIMR - EA 4297
Département de génie des procédés
Université de technologie de Compiègne

Février 2026

Table des matières

1 Pour bien démarrer **5**
- 1.1 Entrer une matrice . . . 6
- 1.2 Les éléments d'une matrice . . . 7
- 1.3 Les commandes Scilab et les variables . . . 7
- 1.4 Obtenir des informations sur l'espace de travail . . . 8
- 1.5 Quitter et sauver l'espace de travail . . . 9
- 1.6 Nombres, expressions arithmétiques et fonctions usuelles . . . 9
- 1.7 Comment obtenir de l'aide . . . 11

2 Les opérations sur les matrices **12**
- 2.1 La transposition . . . 12
- 2.2 Addition et soustraction . . . 13
- 2.3 Multiplication . . . 13
- 2.4 Inversion d'une matrice et division . . . 14
- 2.5 Opérations élément par élément . . . 16
- 2.6 Opérateurs relationnels . . . 17
- 2.7 Opérateurs logiques . . . 18
- 2.8 Fonctions usuelles appliquées à une matrice . . . 18

3 Manipulations des matrices et des vecteurs **20**
- 3.1 Comment générer des matrices et des vecteurs . . . 20
- 3.2 Comment manipuler les éléments d'une matrice . . . 22

4 Scripts et fonctions **25**
- 4.1 Les scripts . . . 25
- 4.2 Les fonctions . . . 27
- 4.3 Contrôle d'exécution . . . 28
- 4.4 Interaction avec l'utilisateur . . . 30

5 Les graphiques **31**
- 5.1 Les graphiques à deux dimensions . . . 31
  - 5.1.1 La commande `plot` . . . 31
  - 5.1.2 Titres et légendes . . . 33
- 5.2 Manipuler plusieurs graphiques . . . 33
- 5.3 Les graphiques à trois dimensions . . . 35

5.3.1 Les courbes 35
5.3.2 Les surfaces 35

6 Import et export de données **38**
6.1 Lecture d'un fichier Excel (.xls) 38
6.2 Lecture et écriture d'un fichier .csv 39

Chapitre 1

Pour bien démarrer

Si vous n'avez pas encore installé Scilab, vous pouvez télécharger la dernière version à l'adresse suivante :

http://www.scilab.org

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
- comment obtenir de l'aide

Si vous avez déjà utilisé Matlab, Python/numpy ou Julia, vous noterez certaines similarités avec Scilab au niveau du langage, à ce titre vous pouvez consulter l'excellente "Cheatsheet" disponible à l'adresse :

https://sciml.github.io/Scientific_Modeling_Cheatsheet/scientific_modeling_cheatsheet

qui permet de passer facilement d'un langage à l'autre. Si vous avez besoin d'aide avec Scilab, vous pouvez aller sur le site communautaire

https://scilab.discourse.group

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

```
Console Scilab 2026.0.1

Initialisation :
  Chargement de l'environnement de travail

 -->
```

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 éventuellement 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 `C` ou Java). 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 = [3x3 double]

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

La matrice `A` est gardée en mémoire pour un usage ultérieur. Notez que les nombres, bien qu'entiers, sont stockés comme des nombres à virgule flottante double précision. D'une manière générale, lorsqu'une matrice n'est pas un scalaire les dimensions et le type sont précisés à l'affichage (pour des scalaires de types particuliers où l'affichage peut être ambigu le type est systématiquement indiqué).

Une 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 = [1x3 double]

  -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 = [1x5 double]

  -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. Par exemple les deux commandes suivantes

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

produisent l'affichage suivant

```
Variable non définie : Z
```

1.4 Obtenir des informations sur l'espace de travail

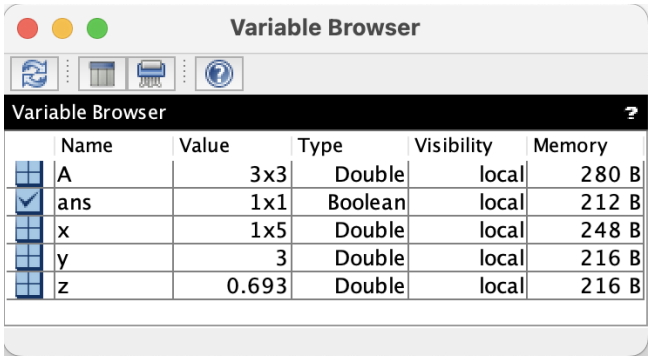


FIGURE 1.1 – Navigateur de variables

Les exemples proposés précédemment ont créé des variables qui sont stockées dans *l'espace de travail* de Scilab. La liste de ces variables est visible dans le *Navigateur de variables* situé à droite de la fenêtre principale de Scilab (voir sur la figure 1.1). On retrouve dans cette fenêtre la liste les 4 variables qui ont été créées dans les exemples précédents.

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.sod")
```

Cette commande sauve toutes les variables dans votre répertoire dans un fichier au format *Scilab Open Data* `work.sod`. Ce format de fichier respecte le format d'échange plus général hdf5 décrit à l'adresse suivante :

`https://fr.wikipedia.org/wiki/Hierarchical_Data_Format)`

Il est possible lors d'une session ultérieure de récupérer le contenu de ce fichier en tapant la commande `load work.sod`.

Il est possible de sauvegarder ou bien de charger uniquement certaines variables. La commande

```
--> save("data.sod","A","x")
```

ne sauve dans ce même fichier que les variables `A` et `x`. La commande `load("data.sod")` permettra de récupérer toutes les variables sauvegardées dans ce fichier (ici `A` et `x`) et `load("data.sod","A")` ne chargera que la matrice `A`.

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 =

   2.220D-16
```

Ce nombre, égal à 2^{-52} et souvent noté ε, est le plus petit nombre positif x tel que l'inégalité $1+x>1$ reste vraie (pour l'ordinateur). Cela signifie que la précision relative est à peu 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. Sinon, et cela n'a rien à voir avec la précision relative évoquée précédemment, 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 =

   0. + 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 =

      3.1415927
```

renvoie la valeur de π, pré-calculée comme étant la valeur de `4*atan(1)`.

1.7 Comment obtenir de l'aide

Pour obtenir de l'aide sur une commande particulière, par exemple ici la commande `max` il suffit de taper

```
--> doc max
```

Les pages d'aide de Scilab proposent des exemples ainsi que des liens vers des commandes similaires ou en relation. Lorsque l'on cherche une commande effectuant une tâche particulière sans connaitre son nom on peut utiliser la commande `apropos` qui cherche les pages d'aide dont le texte contient une chaine de caractères particulière. Par exemple si on désire trier un vecteur ou une matrice on tapera

```
--> apropos tri
```

le navigateur d'aide propose alors une liste de commandes, dont la première est `gsort`, ce qui est une excellente suggestion. D'une manière générale, on peut aussi taper tout simplement

```
--> doc
```

et naviguer dans la table des matières.

Chapitre 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 = [3x3 double]

       1    2    3
       4    5    6
       7    8    0

   B = [3x3 double]

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

et la commande

```
--> x=[-1  0  2]’

   x =   [3x1 double]

      -1.
       0.
       2.
```

Attention, si `Z` est une matrice complexe, alors `Z’` désigne la matrice adjointe (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

Dimensions ligne/colonne incohérentes
```

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 = [3x3 double]

        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 = [3x1 double]

       -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

Dimensions ligne/colonne incohérentes
```

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 = [3x1 double]

         5.
         8.
        -7.
```

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

```
--> A*2

   ans = [3x3 double]

         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  = [3x3 double]

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

--> C=B*A

 C  = [3x3 double]

    1.           4.441D-16    0.
  - 2.220D-16    1.           0.
    0            0            1.
```

On notera au passage les erreurs dues à la précision finie des calculs (la matrice C devrait être égale à la matrice identité).

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 (mathématiquement) 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 $Ay = x$, dont les équations sont

$$\begin{aligned} y_1 + 2y_2 + 3y_3 &= -1, \\ 4y_1 + 5y_2 + 6y_3 &= 0, \\ 7y_1 + 8y_2 &= 2, \end{aligned}$$

on écrira en Scilab

```
--> y=A\x

   y  = [3x1 double]

      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 d'équations avec la méthode d'élimination de Gauss, ce qui est à peu près 3 fois plus rapide que si l'on avait écrit

```
--> y=inv(A)*x

   y  = [3x1 double]

      1.5555556
    - 1.1111111
    - 0.1111111
```

car dans ce cas il faut d'abord calculer l'inverse de la matrice (3 systèmes linéaires à résoudre) puis faire une multiplication matrice-vecteur. On peut vérifier l'exactitude des résultats de la façon suivante :

```
--> A*y-x

 ans  = [3x1 double]

   2.220D-16
   5.551D-16
   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 = [1x3 double]

         4.   10.   18.
```

La division fonctionne de la même manière :

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

   z = [1x3 double]

        0.25   0.4   0.5
```

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

```
--> x.^y

   ans = [1x3 double]

         1.   32.   729.

--> x.^2

   ans = [1x3 double]

         1.    4.    9.

--> 2.^x

   ans = [1x3 double]

         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 = [3x3 double]

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

La commande

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

    P  = [3x3 boolean]

    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 = [3x3 boolean]

    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 = [2x2 double]

      0.    0.25
      0.5   0.75

--> cos(%pi*A)

   ans = [2x2 double]

      1.    0.7071068
      0.  - 0.7071068
```

Chapitre 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 = [3x2 double]

      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 = [3x3 double]

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

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

```
--> x=1:5

   x = [1x5 double]

         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 = [1x5 double]

         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 = [1x6 double]

      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 = [16x2 double]

      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 = [1x5 double]

    - 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 = [3x3 double]

      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 = [3x1 double]

        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 = [1x5 double]

        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 = [3x3 double]

      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 = [3x3 double]

      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 = [1x2 double]

         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.

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 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 :

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

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 `fibo.sce`. La fenêtre créée doit ressembler à celle qui se trouve sur la figure 4.1. Vous pouvez maintenant saisir les

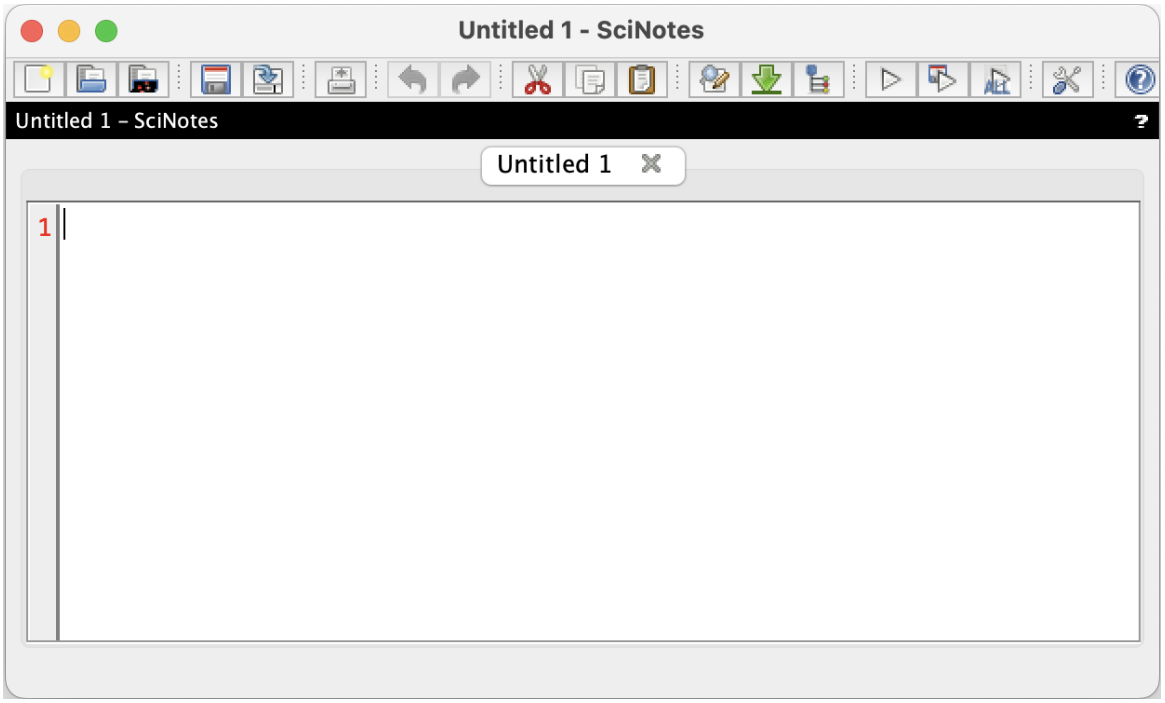


FIGURE 4.1 – Fenêtre de l'éditeur de texte

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

```
u = [10x1 double]

    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 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);
```

Dans le cas où la fonction ne renvoie qu'un seul argument, il est aussi possible de la définir comme une fonction lambda (ce qui est pratique lorsqu'on doit la passer comme argument anonyme à une autre fonction), par exemple :

```
--> plus = #(a,b) -> (1+b)

 plus = function

plus: (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 script. 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 `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
```

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

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

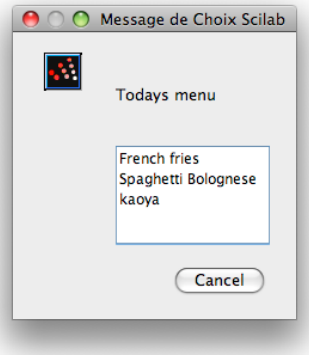


FIGURE 4.2 – Menu obtenu avec la commande `x_choose`

```
    1.
q   =

    3.
```

Notez que quand vous appellez 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 `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 exécutant le script à nouveau grâce au bouton ▷

On peut facilement générer un menu (voir la figure 4.2) 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"],"Todays menu")
```

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 double-clic de souris sur l'une des options, la fonction renvoie le numéro de l'option choisie.

Chapitre 5

Les graphiques

Ce chapitre donne un bref apercu 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))
--> isoview
```

Notez la commande `isoview` 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

```
--> doc plot
```

pour en savoir plus.

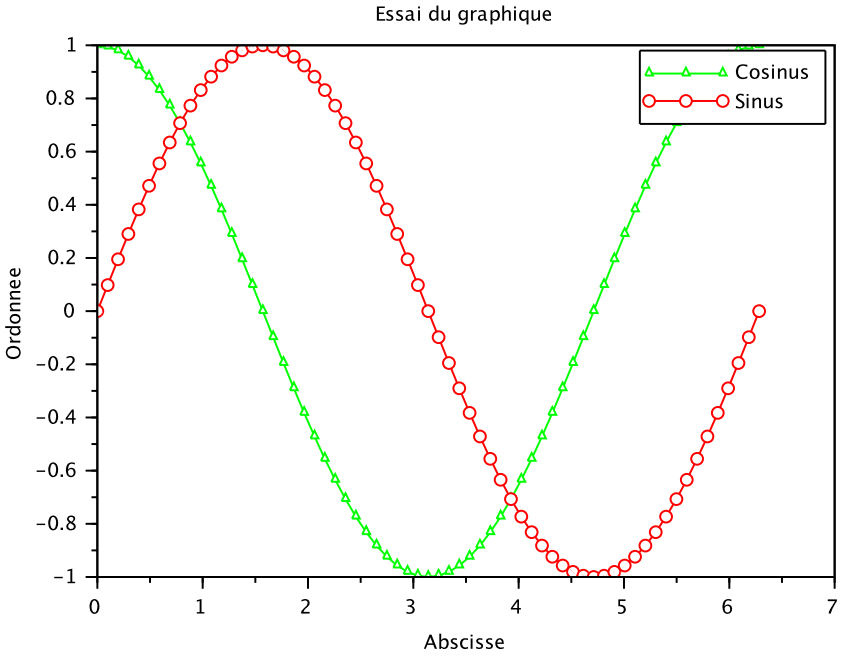


FIGURE 5.1 – Résultat de la commande `plot`

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 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 (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))
```

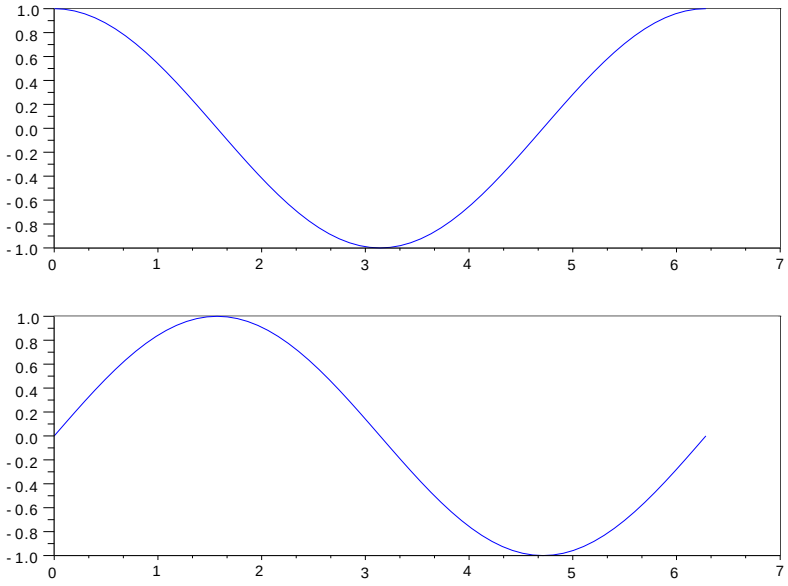


FIGURE 5.2 – Résultat obtenu avec la commande `subplot`

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 :

```
--> 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 20×40. 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 :

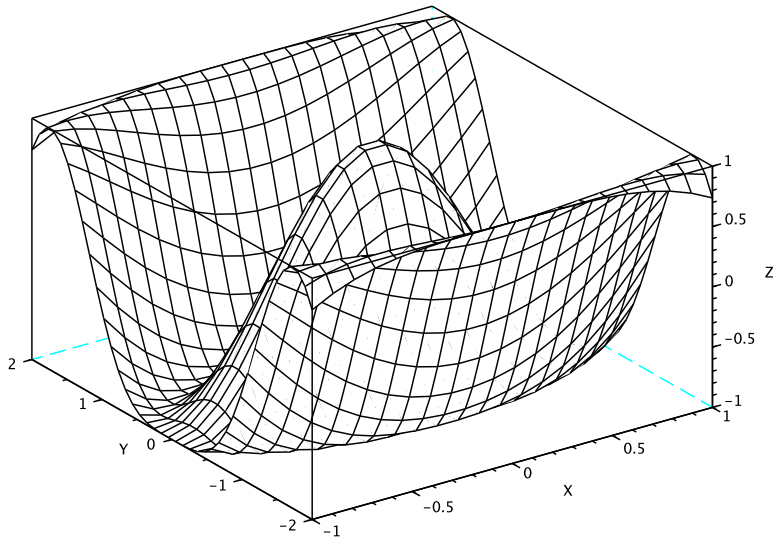


FIGURE 5.3 – Résultat de la commande `mesh`

```
--> [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.

A l’aide des commandes suivantes il est possible de représenter sous forme de couleurs un champ scalaire défini sur la surface :

```
--> clf
--> gcf(),color_map=jetcolormap(128)
--> surf(X,Y,Z)
--> gce().color_flag=3
--> colorbar
```

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 5.4). La commande `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 `gce().color_flag=3` permet d’interpoler les couleurs dans chaque maille. Cela permet d’avoir un aspect “lissé”, plus agréable à l’oeil, sans avoir à prendre plus de points pour générer la surface. Quand à elle, la commande

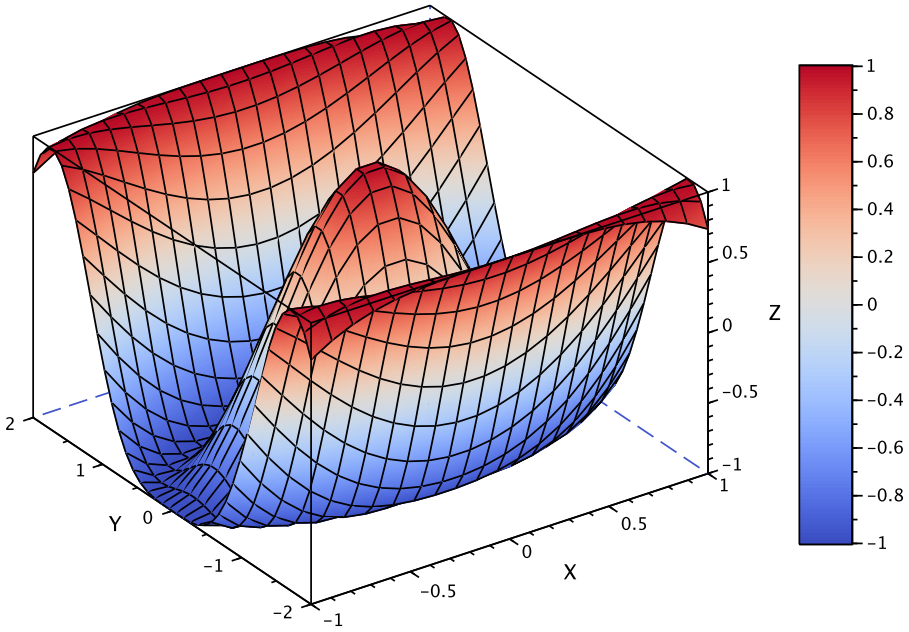


FIGURE 5.4 – Résultat de la commande `surf`

`colorbar` affiche l'échelle des couleurs correspondant au champ scalaire représenté sur la surface (ici la coordonnée z).
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 :

```
--> http_get("http://www.utc.fr/~mottelet/scilab/data.xls","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  = (xls mlist) with fields:

     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 ”structure” avec des champs permettant d'accéder aux feuilles individuelles, sous forme de matrices de chaines 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)")
```

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 au paramètres linguistiques par défaut, en particulier pour la décimale (virgule ou point). On peut par exemple sauver ainsi une matrice aléatoire destinée à être relue dans un tableur dont la langue est réglée en français :

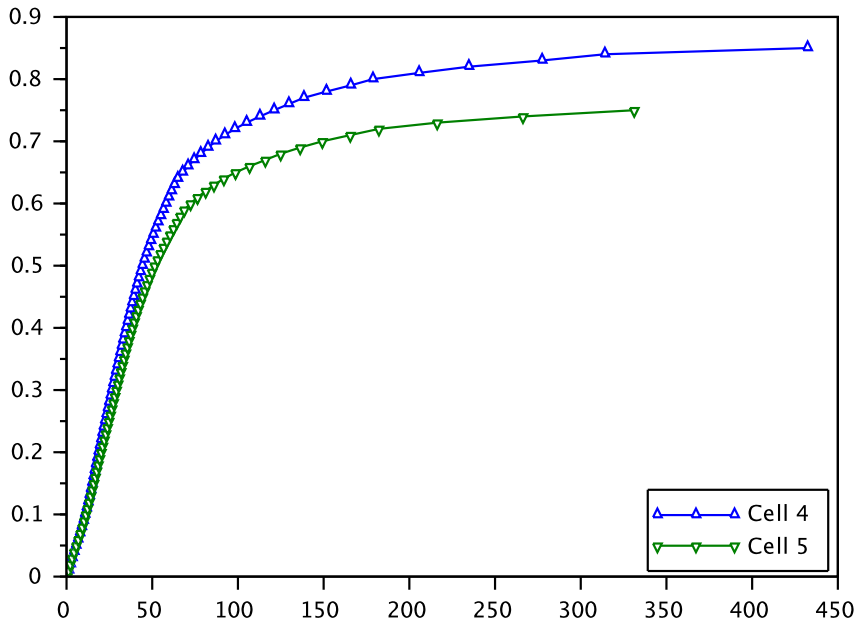


FIGURE 6.1 – Graphique réalisé avec des données extraites d'un fichier Excel

```
--> 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.

Index

π, 11
%eps, 9
%pi, 11

ajouter une ligne à une matrice, 23
ans, 7
apropos, 11

boucles, 28

chiffres significatifs, 10
clf, 35
colormap, 36
commentaires, 26
comparer deux matrices, 17
couleurs, 32
courbe, 31
courbe paramétrique, 31
crochets `[ ]`, 6
csvRead, 39
csvWrite, 39

doc, 11

effacer la fenêtre graphique, 35
else, 27
error, 27
excel, 38
expressions logiques, 18
exécuter un script, 26
eye, 20

floor, 27
for, 28
format CSV, 39
format HDF5, 9
format Scilab Open Data, 9

gca, 31
gce, 36
gcf, 36
gsort, 11

http_get, 38

if, 27
initialiser une matrice, 20
input, 30
inv, 14

legend, 33
linspace, 22
liste des variables, 8
load, 9

matrice identité, 20
matrice nulle, 20
max, 11
mesh, 36
meshgrid, 35

opérateur \, 15
opérateurs usuels, 10

param3d, 35
pdf, 37
permutation, 23
plot, 31
point-virgule, 8
prod, 28
produit scalaire, 14
précision, 10

readxls, 38

sauvegarder, 9
sauver un graphique, 37
save, 9
scalaires, 6
scf, 33
shading, 36
size, 23

sous-matrices, 23
subplot, 34
superposer deux courbes, 32
surf, 36
système linéaire, 15

tableaux, 16
taille d'une matrice, 23
title, 33
tri, 11
type de ligne, 32
télécharger un fichier, 38

variables locales, 27
vecteurs, 6

while, 29

x_choose, 30
xlabel, 33

ylabel, 33

zeros, 20

éditeur de texte, 25