Une (très courte) introduction à Scilab

Table des matières

1 Introduction

Scilab est un pseudo-clone libre de MATLAB développé par l'INRIA (Institut National de Recherche en Informatique et Automatique) qui permet donc d'effectuer des calculs numériques. La syntaxe est très similaire à celle de MATLAB. Il permet, entre bien d'autres fonctionnalités :

— la résolution de système linéaires,
— le calcul de valeurs propres et vecteurs propres,
— la décomposition en valeurs singulières, pseudo-inverse,
— la transformée de Fourier rapide,
— la résolution d'équations différentielles avec plusieurs méthodes,
— d'effectuer des algorithmes d'optimisation,
— la résolution d'équations non-linéaires,
— la création de nombres aléatoires,
— l'utilisation de nombreuses primitives d'algèbre linéaire.

Scilab dispose aussi d'outils permettant de tracer des graphiques (polygônes, images, surfaces, etc...). Il intègre aussi un langage de programmation puissant pour tout ce qui est produit vectoriel/matriciel. Enfin, on peut interfacer le fortran ou le C assez facilement dans Scilab.

Nous ne verrons pas toutes les possiblités de Scilab dans ce tutoriel, mais juste les notions de base pour pouvoir commencer à utiliser cet outil.

2 Expressions scalaires ou matricielles, et affectation

Ce paragraphe a pour but d'apprendre l'utilisation des types de base de Scilab (entiers, réels, complexes, matrices). A part dans la section 5, toutes les commandes sont exécutées dans l'interface de Scilab, que l'on obtient en tapant scilab dans un terminal (ou que l'on exécute si on utilise Windows$^{(tm)}$).

2.1 Affectation

L'affectation sous Scilab est extrèmement simple. Il n'y a pas besoin de déclaration de variable. On ne fait que l'utiliser :

```
variable = expression
```

ou bien

```
expression
```

(pour voir son résultat). Par exemple, pour stocker la valeur 5.2 dans la variable `K`, il suffit d'effectuer :

```
--> K = 5.2
```

Note : Les espaces autour du signe = ne sont pas nécessaires.
Pour avoir le résultat de sin(0.5), il suffit de faire

```
-->sin(0.5)
 ans  =

    0.4794255
```

2.2 Scalaires

Les types scalaires sont les entiers, les réels et les complexes. L'utilisation des entiers ou réels est standard. Par contre, pour contruire un nombre complexe, on utilise la constante %i. Par exemple :

```
-->(1 + %i * 4) * (1 - %i * 4)
 ans  =

    17.
```

2.3 Matrices, vecteurs

2.3.1 Manipulation basique

Un des types de base les plus importants de Scilab est constitué des matrices (les vecteurs étant un cas particulier de matrice. Il est donc à noter qu'une distinction existe entre un vecteur ligne et un vecteur colonne).

Sous Scilab, pour créer une matrice, on utilise la syntaxe suivante :

- les éléments sont des scalaires
- les éléments d'une même ligne sont séparés par une virgule ou un espace,
- chaque ligne, sauf la dernière doit se terminer par un point-virgule,
- la liste des éléments doit être entourée de crochets ouvrant et fermant.

Par exemple :

```
-->[1 1 1; 2 4 8; 3 9 27]
```

produit la sortie

```
 ans =
!  1.   1.   1.  !
!  2.   4.   8.  !
!  3.   9.   27. !
```

Pour garder la valeur de la matrice en mémoire, on peut assigner une variable :

```
-->A = [1 1 1; 2 4 8; 3 9 27]
```

produit la sortie

```
 A =
!  1.   1.   1.  !
!  2.   4.   8.  !
!  3.   9.   27. !
```

Pas besoin de déclaration de variable préalable.

Noter aussi que la matrice doit être correctement définie. En particulier,

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

n'est pas bon, et renverra une erreur, car la dernière ligne est constituée de 2 éléments seulement.

Toute instruction voit son résultat affiché dès que la touche *entrée* a été appuyée. Pour enlever cette fonctionalité, il suffit de rajouter un point-virgule à la fin de l'instruction. Par exemple :

```
-->b = [2 10 44];
```

Pour afficher la valeur d'une variable, il suffit de taper son nom en ligne de commande puis appuyer sur la touche *entrée*. Par exemple :

```
 b =
!  2.   10.   44.  !
```

Pour faire une somme ou un produit matriciel, on utilise la syntaxe habituelle : +, - ou *.

Pour faire une transposée dans le cas réel, ou une transposée-conjugée dans le cas complexe, on utilise l'apostrophe. Par exemple, dans notre cas, `b` est un vecteur ligne. Il nous faudrait un vecteur colonne pour multiplier `A` par `b`. Donc, si on appelle

```
--> A*b
```

on obtient le message

```
-->A*b
    !--error    10
inconsistent multiplication
```

On est donc amené à appeler

```
-->A*b'
 ans  =

!   56.   !
!   396.  !
!   1284. !
```

Ceci met en avant aussi le fait que Scilab vérifie la validité des opérations. Pour faire une somme de deux matrices, celles-ci doivent avoir la même taille. Et pour faire une multiplication de deux matrices, le nombre de colonnes de la première doit être égal au nombre de lignes de la deuxième.

Pour l'accès aux dimensions et au nombre d'éléments d'une matrice ou d'un vecteur on utilise les fonctions `size` et `length` :

- `size(A)` : renvoie un vecteur `[m,n]` où `m` et `n` sont respectivement le nombre de lignes et de colonnes de la matrice (ou du vecteur) `A`. Note : `size(A,1)` renvoie le nombre de lignes `m` et `size(A,2)` renvoie le nombre de colonnes `n`.
- `length(A)` : renvoie le nombre d'éléments de la matrice (ou du vecteur) `A`.

Par exemple :

```
-->A = [1 1 1; 2 4 8 ]; b=[ 0  1 ]; c=b';

-->size(A)
 ans  =

   !  2.   3.  !

-->size(A,1)
 ans  =

     2.

-->size(b)
 ans  =
```

```
   !  1.   2.  !

-->size(c)
 ans  =

   !  2.   1.  !

-->length(A)
 ans  =

     6.

-->length(b)
 ans  =

     2.
```

2.3.2 Accès aux éléments et modification

Pour avoir accès aux éléments d'une matrice ou d'un vecteur, il suffit de d'utiliser les parenthèses ouvrante et fermante et de spécifier les coordonnées, pour les matrices, sous la forme ligne-colonne, et l'indice pour les vecteurs. Par exemple :

```
-->b = [2 10 44];

-->b(1)
 ans  =

    2.
```

et

```
-->A = [1 1 1; 2 4 8; 3 9 27];

-->A(2,3)
 ans  =

    8.
```

ATTENTION : les indices commencent par 1 et non pas par 0, contrairement aux tableaux du C ou du C++. Ainsi, A(1,1) est l'élément haut-gauche de la matrice (restes de fortran).

La modification d'un élément d'une matrice ou d'un vecteur se fait aussi simplement par assignation. Partant de la matrice `A` ci-dessus :

```
-->A(2,3)=-1;

-->A
 A =

   !  1.   1.   1.  !
   !  2.   4.   -1. !
   !  3.   9.   27. !
```

Une fonctionnalité qui peut être pratique (mais qu'il faut utiliser avec parcimonie eu égard à son coût en temps d'exécution et aux accès mémoires nécessaires) est l'extension de matrice. Scilab agrandit automatiquement une matrice pour donner un sens à une assignation. Par exemple :

```
-->A = [1 1 1; 2 4 8; 3 9 27];

-->A(4,5)=2;

-->A
 A =

   !  1.   1.   1.    0.   0. !
   !  2.   4.   8.    0.   0. !
   !  3.   9.   27.   0.   0. !
   !  0.   0.   0.    0.   2. !
```

2.3.3 Matrices et vecteurs types

Il existe des fonctions pour construire des matrices et des vecteurs particuliers. En voici une liste non exhaustive :

matrice identité On fournit le nombre de lignes et de colonnes à la fonction `eye`. Ils peuvent ne pas être égaux.

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

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

matrice diagonale on construit un vecteur `b` et on le passe en paramètre de la fonction `diag` :

```
-->b = [2 10 44];
-->B = diag(b)
 B  =

!   2.    0.     0.  !
!   0.    10.    0.  !
!   0.    0.     44. !
```

Pour récupérer la diagonale, on passe une matrice à la fonction `diag` :

```
-->diag(B)
 ans  =

!   2.  !
!   10. !
!   44. !
```

Pour créer une matrice tri-diagonale, on crée trois vecteurs, correspondant aux trois diagonales, et on utilise la fonction `diag` avec un paramètre supplémentaire, un entier. `diag(b, 1)` construit la matrice ayant pour diagonale supérieure le vecteur `b`. Un entier négatif construit une matrice ayant une diagonale inférieure. Ainsi :

```
-->a = [1 2 3 4];

-->b = [5 6 7];

-->c = [8 9 10];

-->B = diag (a) + diag (b, 1) + diag (c, -1)
 B  =

!   1.    5.    0.     0. !
!   8.    2.    6.     0. !
!   0.    9.    3.     7. !
!   0.    0.    10.    4. !
```

matrice nulle On utilise la fonction `zeros` :

```
-->zeros (4,5)
 ans  =

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

matrice de uns On utilise la fonction `ones` :

```
-->ones (4,5)
 ans  =

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

vecteurs à incrément constant Une très importante fonctionalité de Scilab est la création très rapide de vecteurs à incrément constant, très pratique quand on veut par exemple calculer une discrétisation d'un segment.

Pour créer le vecteur (1,2,3,4,5,6), il suffit d'utiliser la syntaxe :

```
-->1:6
 ans  =

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

c'est-à-dire deux entiers séparés par le symbole deux-point. On peut aussi utiliser

```
-->[1:6]
 ans  =

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

Ainsi, pour créer une discrétisation (ou subdivision) du segment [2,4] avec un incrément de 0.2, on appelle

```
-->X = 2:0.2:4
```

Une autre manière de construire un vecteur à incrément constant est de donner les extrémités du segment et de spécifier le nombre de points (extrémités incluses) que l'on désire. On utilise la commande `linspace`. Par exemple

```
-->linspace(2, 4, 11)
 ans  =

!   2.    2.2    2.4    2.6    2.8    3.    3.2    3.4    3.6    3.8    4. !
```

2.3.4 Extraction de sous-matrice et de sous-vecteurs

Il est très facile sous Scilab d'extraire des sous-matrices et des sous-vecteurs. Le principe est le même que l'accès aux éléments, généralisé en mentionnant les différents indices de lignes et/ou de colonnes qu'on veut extraire sous forme de vecteurs d'indices. Par exemple, en utilisant la dernière matrice `A` obtenue ci-dessus :

```
-->A(3,[1 3 4]) // Matrice extraite de A composé des éléments
                  // sur la 3ème ligne et les colonnes 1, 3 et 4.
ans =

   !  3.   27.   0. !

-->A([1 2],[1 2]) // Matrice extraite de A composée des éléments
           //sur les deux premières lignes et deux premières colonnes.
ans =

   !  1.   1. !
   !  2.   4. !

-->A(2:2:4,1:2:5) // Matrice extraite de A composée des éléments
                  //sur les lignes paires et les colonnes impaires.
ans=

   !  2.    8.    0. !
   !  0.    0.    2. !

-->A(2,:) // Deuxième ligne et toutes les colonnes
                  // (sans préciser les indices)
ans=

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

De même, on peut modifier toute une partie de la matrice directement avec l'assignation d'une sous-matrice :

```
-->A(1,:)=[-1 -1 -1 -1 -1] // remplacement de la première ligne
 A =

   ! -1.  -1.   -1.  -1.  -1. !
   !  2.   4.    8.   0.   0. !
   !  3.   9.   27.   0.   0. !
   !  0.   0.    0.   0.   2. !

-->A(3:4,3:5)=[1 1 1 ; 1 1 1]
 A =

   ! -1.  -1.  -1.  -1.  -1. !
   !  2.   4.   8.   0.   0. !
   !  3.   9.   1.   1.   1. !
   !  0.   0.   1.   1.   1. !
```

Attention cependant, dans ce cas, les sous-matrices doivent être correctement définies, i.e. l'extension ne fonctionne plus :

```
-->A(1,:)=[0 0 0 0 0 0]

                  !--error 15
Sous-matrice incorrectement définie.
```

2.4 Matrices et fonctions usuelles

Les fonctions usuelles peuvent, dans Scilab, aussi bien s'appliquer sur des scalaires que sur des vecteurs, voire des matrices. Par exemple, pour avoir la liste des valeurs de $\sin(x)$ pour $x = 0, 0.1, 0.2, \ldots, 1$, il suffit d'appeler :

```
-->sin(0:0.1:1)'
 ans  =

!   0.        !
!   0.0998334 !
!   0.1986693 !
!   0.2955202 !
!   0.3894183 !
!   0.4794255 !
!   0.5646425 !
!   0.6442177 !
```

```
!   0.7173561 !
!   0.7833269 !
!   0.8414710 !
```

(noter la transposition pour l'affichage en colonne). Ainsi, la fonction sin a été appliquée à toutes les composantes du vecteur $[0 : 0.1 : 1]$.

La liste des fonctions usuelles peut être trouvée dans l'aide de Scilab.

2.5 Opération "élément par élément"

Dans la perspective d'appliquer la même fonction aux éléments d'un vecteur ou d'une matrice, il faut distinguer correctement la multiplication de matrice et la multiplication élément par éléments de deux matrices. Pour multiplier ou diviser deux matrices élément par élément, il faut rajouter un '.' devant l'opérateur :

```
-->A = ones(3,3);

-->B = 2 * ones(3,3);

-->A.*B
 ans  =

!   2.    2.    2. !
!   2.    2.    2. !
!   2.    2.    2. !

-->A./B
 ans  =

!   0.5    0.5    0.5 !
!   0.5    0.5    0.5 !
!   0.5    0.5    0.5 !
```

De même, pour la transposition, lorsque qu'on dispose d'une matrice complexe et qu'on ne veut que transposer, sans prendre le conjugué de la matrice :

```
-->A=ones(2,2) + %i * ones(2,2);

 -->A(2,1) = 0.0;

-->A'
```

```
 ans  =

!   1. - i      0        !
!   1. - i      1. - i   !

-->A.'
 ans  =

!   1. + i      0        !
!   1. + i      1. + i   !
```

3 Visualisation de graphes

Supposons que l'on veuille visualiser la courbe de la fonction $f(x) = e^{-x}\sin(4x)$, pour $x \in [0, 2\pi]$. On commence par créer une discrétisation (ou subdivision) X de l'intervalle $[0, 2\pi]$ et on fait la multiplication élément par élément de $exp(-X)$ et de $sin(4 * X)$:

```
-->X = 2*%pi*[0:0.01:1];

-->Y = exp(-X).*sin(4*X);
```

Enfin, on appelle la fonction d'affichage `plot` :

```
-->plot(X, Y)
```

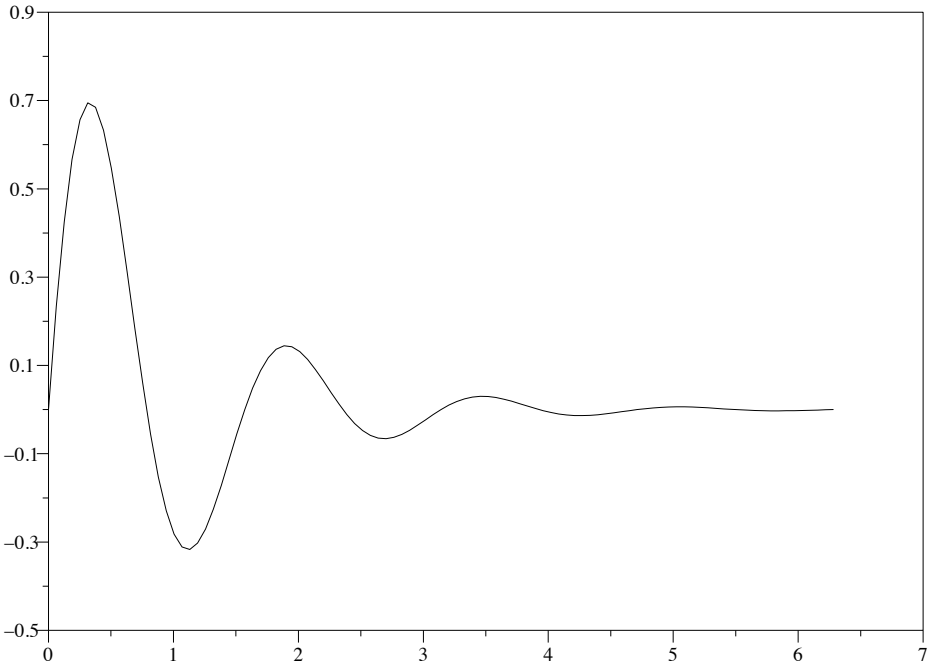
On obtient le graphe suivant.

On remarque que Scilab trace des segments reliant les points voisins dans la discrétisation de la courbe pour en donner une approximation. Plus la discrétisation est fine (i.e. plus elle contient de point), plus l'approximation de la courbe sera précise.

Note : une série d'options peut être utilisée à des fins cosmétiques : préciser le nom des axes, afficher une légende, etc. En outre, il existe d'autres routines pour afficher des graphiques, en 2D, en 3D, avec des possibilités de zoom. On consultera l'aide de Scilab pour en connaître le principe d'utilisation.

4 Programmation en Scilab

En dehors des instructions toutes prêtes, Scilab dispose d'un langage de programmation simple, mais assez complet. On peut faire des boucles, des tests, créer des fonctions, etc... La principale diffférence par rapport aux



langages classiques (tels le C, C++, fortran, etc...), et que nous avons déjà aperçu, est que l'on ne déclare pas les variables. Ceci offre une grande souplesse.

4.1 Les boucles

Les boucles sont à la base de l'algorithmique classique. De manière générale, l'imbrication de plusieurs niveau de boucles est à éviter en Scilab en raison de la lenteur d'exécution. On peut très souvent remplacer certaines boucles par l'utilisation judicieuse d'un vecteur comme nous le verrons plus tard.

Il existe deux types de boucles : la boucle `for` et la boucle `while`.

4.1.1 La boucle `for`

La boucle `for` itère sur les composantes d'un vecteur ligne :

```
-->v = [1 -1 1 -1];

-->y = 0; for k = v, y = y + k, end
 y  =
```

```
    1.
 y  =

    0.
 y  =

    1.
 y  =

    0.
```

Ici, k prend successivement toutes les valeurs de v et on itère sur ces valeurs.

Une autre utilisation classique, produisant le même résultat que la boucle ci-dessus :

```
-->y = 0; for k=1:4, y = y + v(k), end
```

4.1.2 La boucle `while`

Elle permet de répéter une suite d'instruction tant qu'une condition est vraie :

```
-->x = 1; while x < 14, x = 2 * x, end
 x  =

    2.
 x  =

    4.
 x  =

    8.
 x  =

    16.
```

4.2 Les instructions conditionnelles

Il s'agit d'un test d'une ou plusieurs conditions.

Un exemple avec une condition :

```
-->x = 4; if x>0 then, y=-x, else, y = x, end
 y  =
```

```
 - 4.
```

Un deuxième exemple avec plusieurs conditions et plusieurs résultats associés :

```
-->x = 4; if x == 0 then, y = 0, elseif x < 0 then, y = -1, ...
else y = 1, end
 y  =

 - 1.
```

Remarquez également l'utilisation de "==" pour le test d'égalité pour le différencier du "=" de l'assignation. Pour information, en Scilab, on code

- $\leq$ par <=,
- $\geq$ par >=,
- $\neq$ par <>.

5 Ecrire et exécuter un script

Lorsqu'on veut implémenter un algorithme nécessitante plusieurs lignes de commande, il devient vite difficile et lourd d'utiliser directement l'interface de Scilab. On peut donc écrire un script (au même titre qu'on écrit un programme en C++ ou fortran) contenant les diverses commandes Scilab, et exécuter ce script.

5.1 Ecrire un script

Pour cela, il suffit d'ouvrir un fichier (ayant, en général l'extension '.sce') et d'y mettre le code que l'on aurait mis dans l'interface Scilab. A la différence près qu'on peut arranger le programme sur plusieurs lignes, ce qui le rend plus lisible. Il est également à noter que le code est alors réutilisable. Par exemple :

```
// Mon premier programme Scilab

a = 0; b = 1; n = 100;

// calcul des abscisses
x = linspace (a, b, n+1);

// calcul des ordonnees
```

```
y = exp(-x).*sin(4*x);

// ouverture d'une nouvelle fenêtre graphique
scf();

// tracé du graphe
plot (x, y);
```

Plusieurs remarques :

- les commentaires dans un script Scilab sont ceux du C++ : //
- les extensions des scripts sont en général '.sce'. Lorsque le script ne contient que des fonctions, il devient '.sci'.

5.2 Exécuter un script

Enfin, il faut bien exécuter ce script. Supposons que le script ci-dessus soit sauvegardé dans un fichier nommé **mon_script.sce** et qu'il soit dans le répertoire d'où a été lancé Scilab, alors, il suffit de taper :

```
-->exec('mon_script.sce');
```

S'il n'est pas dans le même répertoire, il faut spécifier le chemin.

On peut aussi utiliser le menu 'File' et sélectionner le script en question.

5.3 Ecriture de fonctions

On peut aussi écrire des fonctions, qui seront appelées dans un ou des scripts. On les sauvegarde dans des fichiers ayant l'extension '.sci'.

On commence par un exemple :

```
// fonction  : 'ma_fonction'
// parametre : x, un vecteur de reel
// retour    : A, un vecteur de reel

function A = ma_fonction (x)
  A = exp(-x) .* sin (4 * x)
endfunction
```

La syntaxe est là encore très souple :

- L'implémentation d'une fonction commence par une ligne de déclaration contenant dans l'ordre :
  - le mot clé `function` suivi d'un espace,

  - le vecteur des *sorties* de la fonction (ce qu'on veut évaluer en sortie)
  - le signe "=",
  - le nom de la fonction
  - les *entrées* entre parenthèse, séparés par une virgule.
- Là encore, on ne spécifie pas de type pour les entrées.
- On écrit ensuite les lignes de code.
- On termine par le mot clé `endfunction`
- Noter que dans le cas où une valeur est retournée, on la spécifie devant le nom de la fonction (comme dans l'exemple ci-dessus).

L'écriture des codes est également largement facilité car on n'a essentiellement plus besoin de virgules. Ainsi le deuxième exemple de la section 4.2 pourrait être traduit dans la fonction suivante :

```
function y =  signe(x)
    if x == 0
        y=0
    elseif x<0
        y = -1
    else
        y=1
    end
endfunction
```

On remarquera également qu'en utilisant correctement les passages à la ligne pour un test, il n'est plus nécessaire d'utiliser le mot-clé "then".

6 Quelques fonctions Scilab courantes

Variables et assignations :

- `clear` : élimine toutes les assignations de variables

Graphiques :

- `scf()` : ouvre une nouvelle fenêtre graphique.
- `scf(k)` : ouvre ou sélectionne la fenêtre graphique n^o`k`.
- `gcf()` : efface tous les éléments de la fenêtre graphique courante.
- `gcf(k)` : efface tous les éléments de la fenêtre graphique n^o`k`.
- `xdel()` ou `close` : ferme la fenêtre graphique courante.
- `xdel(k)` ou `close k` : ferme la fenêtre graphique n^o`k`.
- `xdel(winsid())` : ferme toutes les fenêtres graphiques.