Une introduction à Scilab

Scilab 5.4.1 est un logiciel libre, téléchargeable à l'adresse : `http://www.scilab.org/`. Une documentation assez détaillée (114 pages) est téléchargeable à l'adresse :

`http://www.iecn.u-nancy.fr/~pincon/scilab/docA4.pdf`

ici `http://www.scilab.org/content/download/849/7897/file/Scilab_debutant.pdf` il y a un autre texte.

Scilab est un logiciel de calcul scientifique développé par l'INRIA et l'ENPC depuis 1990. Comme Matlab (Matrix Laboratory), Scilab est basé sur la philosophie selon laquelle toute expérimentation numérique peut s'écrire et être programmée à l'aide du calcul matriciel. Scilab est un langage interprété, donc a priori moins performant qu'un langage compilé comme Fortran ou C. Mais Scilab est également un langage évolué. Cela permet d'éviter la programmation de toutes les routines de base, ce qui le rend un outil indispensable dans les domaines de la simulation et de la modélisation.

Le logiciel Scilab accepte des commandes sous deux formes possibles :
– de facon interactive au clavier dans la fenêtre de base (`Console Scilab`)
– sous la forme d'une ou plusieures fonctions écrites dans un ou des fichiers de commande (fichiers scripts - nom-de-fichier.**sce**).

Dans la barre de la fenêtre de commandes (`Console Scilab`)
– Le bouton `?` permet de lancer l'Aide de Scilab. Ensuite, la Loupe permet d'obtenir de l'aide sur une commande donnée. A utiliser aussi souvent que possible.
– Le premier bouton de la barre supérieure ("Démarrer SciNotes") permet de lancer l'éditeur dans lequel vous écrirez vos scripts. On peut aussi lancer l'éditeur en ouvrant un fichier existant à l'aide du deuxième bouton ("Ouvrir un fichier"). Remarque : avant de lancer l'éditeur, vous avez intérêt à vous placer dans le répertoire où vous comptez sauvegarder vos fichiers, à l'aide du sixième bouton de la barre supérieure ("Changer le Répertoire Courant").
– L'avant-dernier bouton ("Démonstrations Scilab") permet d'exécuter une démonstration de Scilab (conseillé pour se rendre compte des possibilités du logiciel).

Les flèches tapées dans la fenêtre de commande permettent de se déplacer dans l'historique des instructions précédentes. Dans un fichier, la suite de symboles `//` met une ligne en commentaire. Un point-virgule `;` permet de supprimer l'affichage du résultat. Scilab distingue les minuscules et majuscules.

Pour commencer un fichier script, on commencera toujours par effacer les variables en mémoire et les figures existantes afin d'éviter les mauvaises surprises :

`clear`	re-initialise toutes les variables
`xdel(winsid())`	ferme toutes les fenêtres graphiques

1. Premières commandes

`2+3; 2*3; 2^3; 2**3`	Les opérations élémentaires
`2+3`	Quelques commandes à essayer
`2+3;`	
`2+3; 3+4`	
`2+3, 3+4`	
`sqrt(-2)`	Scilab connaît les nombres complexes
`a=1, clear a, a`	La fonction clear désaffecte la variable a
`sin(2), abs(-2), imag(2+%i),`	Les fonctions élémentaires
`help eye`	Aide en ligne
`help matrice`	

2. Construction de vecteurs et de matrices ; opérations de base

`x=2:6`	Différents vecteurs
`0:2:8`	
`[0:2:8]`	
`[5,3,2]`	
`linspace(0,20,5)`	Vecteur de 5 points équidistribués entre 0 et 20 (voir aussi `logspace`)
`A=[2,3;4,5;6,7]`	Une matrice de type (3,2) (lignes séparées par des ;)
`A=[2 3;4 5;6 7]`	la même chose
`B=zeros(5,20)`	Matrice de zéros
`C=ones(2,5)`	Matrice de 1
`D=eye(5)`	Matrice identité d'ordre 5
`diag([0:2:8])`	Matrice diagonale
`rand(3,4)`	Matrice de taille (3,4) de nombres aléatoires compris entre 0 et 1
`size(B)`	Dimensions de B
`size(B,1), size(B,2)`	Nombre de lignes, nombre de colonnes
`length(x)`	Taille de x
`A'`	Transposée
`C+D, E=A*A', inv(E)`	Addition, produit et inverse matriciels
`trace(E)`	Trace de E
`spec(E), rank(E), det(E)`	Valeurs propres, rang, déterminant
`clean(spec(E))`	Arrondit à zéro les nombres inférieurs en valeur absolue à 10^{-10}
`cond(E)`	Conditionnement
`norm(E,1), norm(E,'inf')`	Différentes normes matricielles
`norm(E,'fro')`	
`triu(C)`	Extrait la partie triangulaire supérieure de C
`tril(C)`	Extrait la partie triangulaire inférieure de C
`diag(C)`	Extrait la partie diagonale de C

3. Les éléments et les sous-matrices

`E(1,2)`	l'élément E_{12}
`E(:,3), E(3,:)`	troisième colonne de E, troisième ligne de E
`E(:,1:2)`	les deux premières colonnes
`E(:, $ )`	la dernière colonne de E
`E([1,2],[2,3])`	sous–matrice de taille $2\times$ 2
`diag(E)`	vecteur colonne des éléments diagonaux de E
`diag(diag(E))`	
`E([2,1,3],:)`	permutation des 2 premières lignes de E

`[A, B]`, pour `A` une matrice $m \times n$, et `B` une matrice $m \times p$, est la matrice bloc $m \times (n+p)$ dont les n premières colonnes sont celles de `A` et les p suivantes celles de `B`. On peut aussi utiliser `[A B]`, sans la virgule. `[A; B]`, pour `A` une matrice $m \times p$, et `B` une matrice $n \times p$, est la matrice bloc $(m+n) \times p$ dont les m premières lignes sont celles de `A` et les n suivantes celles de `B`.

4. D'autres opérations sur les matrices

Certaines opérations s'appliquent sur les matrices (par ex. la multiplication si les dimensions sont convenables) et certaines opérations n'ont pas de sens (par ex. la division). Si on veut appliquer ces opérations terme à terme, on les précède par un point. Voici quelques exemples.

```
v=[1,2,3]
v/2
v./2
w=[1,2,3]
v./w
v./(1:1:3)
v./(1:3)
v(1:1:3)./(1:1:3)
(0:0.1:1).^2
```

`v/w`	Cette division n'a pas de sens mathématique. Cependant, en Scilab, cette commande résout l'équation $x * w = v$. Ce sont les dimensions de v et w qui déterminent les dimensions de la réponse x. Le système peut être sous- ou sur-déterminé.
`[1,2,3]/[1,3]`	
`[1,2]/[1,1]`	L'équation à résoudre n'a pas de solution. Pourtant Scilab renvoie un resultat. Une idée de ce que représente la réponse ? Faire attention à l'utilisation de cette commande...
`w\v`	résout l'équation $w * x = v$
`E+1`	Une addition possible en Scilab
`sin(%pi*[0,1;1,0])`	Les fonctions élémentaires s'appliquent aussi aux matrices
`abs(E), imag(E+%i)`	
`[m,k] = max(E)`	Qu'obtient–on dans les variables m et k ?

5. Les graphiques 2D

La représentation d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ commence par la création d'un vecteur d'abscisses, auquel on applique la fonction pour créer le vecteur des ordonnées.

`x=linspace(0,2*%pi,100)`	(essayez aussi 5 à la place de 100)
`plot(sin(x))`	Trace les pts d'ordonnées `sin(x)` correspondant aux abscisses `(1:size(x,))2`
`plot(x,sin(x))`	Trace les pts d'ordonnées `sin(x)` correspondant aux abscisses `x`
`plot(x,[sin(x);sin(2x)])`	On peut aussi tracer deux graphes
`clf`	On efface le contenu dans la fenêtre de plot

Il est possible d'ajouter des options pour le tracé : `plot(x,y,':or')` : le symbole `':'` pour relier les points par des traits en pointillées, le `'o'` pour repérer les points `(x(i); y(i))` par un petit rond et le `'r'` pour un tracé en rouge. Voir l'aide dans LineSpec pour les options possibles. Avec plot, on peut éegalement tracer plusieurs courbes dans le même repère : `plot(x,sin(x),'o-r',x,sin(2*x),'*-b')`

Pour améliorer son dessin, on peut ajouter divers objets (on donne parfois deux syntaxes alternatives, et on suppose d'avoir tracé deux courbes)

`figure(7)`		on trace dans la fenêtre d'indice 7
`xdel(7)`		on ferme cette fenêtre d'indice 7
`legend('x->sin(x)','x->cos(x)')`		des explications
	`a=gca()`	après `plot` il faut initialiser le pointer
`xtitle('le titre')`	`a.title='le titre';`	on donne un titre au dessin
`xlabel('abcsi')`	`a.x_label='absci';`	on donne un titre à l'axe des abscisses
`ylabel('to')`	`a.y_label='to';`	on donne un titre à l'axe des ordonnées
	`a.log_flags='ln';`	l'axe des abscisses en échelle logarithmique
	`a.log_flags='nl';`	l'axe des ordonnées en échelle logarithmique
	`a.log_flags='ll';`	les deux en échelle logarithmique
	`a.isoview='on'`	ou `'off'` : même échelle sur les deux axes
	`a.filled='on'`	ou `'off'` : couleur du fond
	`a.grid=[1,1];`	mettre un maillage abscisses/ordonnéees constantes

Aussi, on peut spécifier par `a.data_bounds=[xmin,ymin;xmax,ymax];` que l'on trace seulement les abscisses dans l'intervalle $[xmin, xmax]$ et les ordonnées dans $[ymin, ymax]$. Finalement, la commande `subplot(3,2,5);` permet de tracer plusieurs repères dans la même figure : ici en total 6 (3 lignes et 2 colonnes), et on se place dans le repère no 5 (en comptant de gauche à droite).

6. Les structures de contrôle et les opérateurs logiques

`for i=1:n,v(i)=1/i^2;end`	Boucle `for` ; la boucle peut être interrompue par une instruction `break` (à éviter)
`a=10, while a > 1, a=a/2; end; a`	Boucle `while`
`clear a, i=1, j=2`	
`if i==j then a(i,j)=2;`	Exécution conditionnelle `if then else`
`else a(i,j)=3; end,`	

`%t, %f`	les variables booléennes
`1==1 & 0==1, 1 & 2,`	Le `et`, les arguments peuvent être matriciels
`1 | 0`	Le `ou`, les arguments peuvent être matriciels
`~[%t %t %f]`	La négation (avec un argument matriciel)

Pensez à rendre votre code lisible, en utilisant des espaces/tabulateurs

```
for i =1:10
    x(i)=i**2;
    y(i)=x(i )+1;
end
```

Pour être efficace, il faudra éviter tant que possible les boucles `for` et écrire des formules vectorielles, ici `x=[1:10].**2; y=x+ones(size(x));` ou `x=[1:10].**2; y=x+1;`

7. Les fonctions

En utilisant l'éditeur de Scilab, on écrit dans un fichier `fonctions.sci` une ou plusieurs fonctions implémentant les suites d'instructions réalisant une tâche - les algorithmes - dont on a besoin. Le fichier pourra ensuite être chargé dans la session Scilab avec la commande `exec fonctions.sci`.

La syntaxe d'une fonction est la suivante :

```
function [x,y]=myfct(a,b)
\\ suite d instructions
x=a+b, y=a-b
z=sin(a),     passe la valeur d'une variable locale dans la session Scilab
pause         suspend l'exécution de la fonction, elle reprendra avec resume
endfunction
```

Pour exécuter la fonction, on tape dans la fenêtre de commande :

`myfct(1,2)`	renvoie la valeur de x
`[x,y] = myfct(1,2)`	renvoie les valeurs de x et y

Pour éviter d'avoir à trop naviguer dans l'historique avec les flèches, il est fortement conseillé d'écrire les commandes dans un fichier `script.sce` à l'aide de l'éditeur :

`clear`	efface toutes les variables locales
`exec fonctions.sci`	charge les fonctions sauvées dans `fonctions.sci`
`n=1, m=2`	
`[x,y] = myfct(n,m)`	

Pour définir une fonction simple, on peut utiliser la commande `deff` dans le fichier script :

```
deff('y=f(x)','y=x.*sin(x)')
deff('y=g(x)','y=ones(x)./sqrt(x)')
```

Scilab connait les fonctions élémentaires `sin asin cos acos tan atan sinh asinh cosh acosh tanh atanh exp log log10 log2 sqrt abs round fix` ainsi que pour les arguments vectorielles/matricielles `max min sum prod`.

8. Les entrées sorties

Quand on fait du calcul matriciel, les données sont vite nombreuses et donc pour être efficace, on utilise des fichiers de données et de résultats.

Pour la lecture : `exec donnees.sce`

Pour l'affichage l'écran :

`disp('c''est exo 1');`	affiche texte à l'écran
`disp(A)`	affiche à l'écran la matrice A (prend beaucoup de place)

Il y a aussi la commande plus élaborée `printf(format,A,B,..,E)` ou A etc sont les variables (scalaires ou vecteurs colonne de même taille) ou alors des chaînes de caractères, et format est une chaîne de caractères comportant `\n` (passage à la ligne), `%s` ou `%3s` pour des chaînes de caractères (à 3 caractères), `%d` ou `%4d` pour des entiers (à 4 chiffres), `%f` ou `%7f` pour des nombres décimaux (à 7 chiffres), `%é`ou `%11.2e` pour des nombres à virgule flottante (à 11 chiffres, 2 après la virgule), et `%` remplace une variable (dans l'ordre), par exemple

```
i=2;b=%pi;printf('\n %s%7d%6s%5f','ecrivons ',i,'fois',b);
i=2;b=%pi;printf('\n %s%7d%6s%11.3e','ecrivons ',i,'fois',b);
```

Pour l'écriture dans un fichier :

```
A=[1 2 3 ; 4 5 6 ; 7 8 9]
B=[1 0 ; 0 1]
```

`write('fichier.dat',A)`	Sauvegarde de la variables A dans le fichier 'fichier.dat' (dans le format Scilab lisible par l'utilisateur)
`A=read('fichier.dat',3,3)`	Lecture de la matrice A
`C=read('fichier.dat',2,2)`	Lecture de la sous-matrice principale 2*2 issue de A
`D=read('fichier.dat',-1,2)`	Lecture de la matrice A (nb de lignes non spécifié)
`save('result',A,B)`	Sauvegarde (en binaire) des variables A et B dans le fichier `result`
`clear A, clear B`	
`load('result','A','B')`	Chargement dans la session des variables sauvegardées dans le fichier `result`