

Document d'initiation au logiciel SCILAB

O. Bokanowski

Janvier 2007

1 Introduction

"SCILAB", abréviation de "**Scientific Laboratory**", est un logiciel de calcul scientifique développé par l'Inria. Il est téléchargeable gratuitement à l'adresse <http://www.scilab.org>. Nous travaillerons avec la version SCILAB-4.1.

SCILAB est particulièrement adapté au calcul numérique matriciel, possède un certain nombre de fonctions graphiques, et est relativement simple d'utilisation.

Ce logiciel "freeware" est équivalent au logiciel MATLAB pour les fonctions de base.

Ouvrir une fenêtre de commande **Xterm**. Dans cette fenêtre, tapez

```
scilab
```

Une autre fenetre apparait, avec un prompt `-->` qui attend vos instructions. Pour quitter Scilab, tapez `quit`, ou utilisez la souris : cliquez sur les boutons jaunes `file` puis `quit`.

2 Le calcul matriciel en SCILAB

En Scilab, tout est matrice, c'est à dire que la variable de base est une matrice.

Dans Scilab, essayez successivement les instructions suivantes :

```
a
a=1
a=1;
2.1+a
b=3*ans
```

Voici ce que vous devez obtenir (sans les commentaires `// ...`)

```
--> a                // . Renvoie un message d'erreur car la variable "a"
                    //   n'est pas encore definie.

!--error      4
undefined variable : a

--> a=1              // . Cree la variable nommee "a" et
                    //   affecte la valeur 1 a la variable "a"

a =

1.
```

```

--> a=1;           // Idem, mais sans affichage.

--> a              // Pour afficher la valeur de a
a =

1.

--> 2.1+a          // . Remarquez que le dernier resultat est appelee' "ans"
                  // a defaut de lui avoir donne' un nom.
ans =

3.1

--> b=3*ans
b =

9.3

```

Separateurs d'instructions , et ;

La virgule , permet d'avoir plusieurs instructions sur la meme ligne. Le point-virgule ; joue le même role, sauf que le résultat n'est pas affiché. Par exemple dans

```
a=1; b=a+1, c=0;
```

seul le résultat de `b=a+1` est affiché.

Interface SCILAB : touches de déplacement

On peut revenir sur une commande précédente, et la modifier eventuellement, en utilisant les flèches du clavier ($\uparrow$, $\downarrow$, $\leftarrow$, $\rightarrow$). Par exemple, revenir sur l'instruction precedente et la modifier en :

```
a=0; b=a+1, c=0;
```

(Note : on peut taper sur "Return" même si le curseur n'est pas en fin de ligne).

Premier pas en calcul matriciel.

Dans la suite, nous utiliserons souvent la présentation suivante :

```

INSTRUCTIONS  SCILAB          //      COMMENTAIRES;
(A TAPER AU CLAVIER)          //      ET/OU QUESTIONS (A COMPRENDRE !)
```

Essayez les instructions suivantes :

```

A=[ 1 2; 1 3; 1 4] // Definir une  matrice.
A=[1 2 3]
A=[1,2,3]           // Idem
A=[1 2; ..
  1 3]              // Le .. permet le passage a la ligne
A(2,2)              // Lecture
A(2,2)=%i           // Affectation (%i= sqrt(-1) variable predefinie)
```

```

A'           // conjuguee (=transposee dans le cas reel)
A.'          // transposee
B=[2 2; 2 2]; A+B // Addition matricielle
A*B          // Multiplication matricielle
A.*B         // Multiplication matricielle terme a terme
A./B         // Division terme a terme
C=[1 2], A+C  // Erreur car dimensions incompatibles

```

On retiendra notamment que pour la creation de matrice par blocs on a les operateurs de concaténation `[ , ]` et `[ ; ]` :

<code>C=[A,B], C=[A B]</code>	concaténation $C=[A \ B]$
<code>C=[A;B]</code>	concaténation $C=\begin{pmatrix} A \\ B \end{pmatrix}$
<code>A'</code>	Matrice conjuguée (matrice transposée dans le cas réel)
<code>A.'</code>	Matrice transposée

Voici quelques exemples :

```

v=[1 2 3]', w=v+ones(v)
[v,w]
x=[v;w]
y=[v v w]

```

Essayez ces autres commandes standards (Un resumé est fourni en annexe).

```

zeros(3,3)      // matrice de zeros
A=rand(3,3)     // matrice de nombre aleatoires (loi uniforme U(0,1))
zeros(A)        // matrice de zeros de meme taille que A
ones(A)         // matrice de 1 de mem taille que A
B=eye(A)        // matrice identite de meme taille que A
x=diag(A)       // extraction du vecteur diagonale de A
D=diag(x)       // matrice diagonale de composantes x(i)
diag(x,1)       // matrice avec la liere diagonale de composantes x(i)
diag(x,-1)      // matrice avec la liere sous diagonale de composantes x(i)
diag(diag(A))   // matrice diag(A_11,...,A_nn)
A + 2*ones(A)   // Ajouter 2 a chaque element de A
A + 2           // Idem
size(A)         // taille de A
size(A,1)       // nombre de lignes
size(A,2)       // nombre de colonnes

```

Aide en ligne. Les deux commandes d'aide à connaître sont `help` et `apropos`. La commande `help` s'utilise si on connaît le nom de la fonction. Par exemple pour une aide sur `diag` on peut taper

```
help diag
```

Si on ne connaît pas le nom de la commande, on peut faire une recherche par mot-clé à l'aide de la commande `apropos`. Par exemple :

```
apropos diagonal
```

(puis cliquer sur le nom de la commande qui vous semble correspondre à votre recherche).

On pourra aussi cliquer dans la fenetre Scilab sur les boutons **Demos** ou **Help...**

Et les vecteurs ?

Ce sont des matrices lignes ou colonnes en SCILAB. Pour la création de vecteurs on utilise souvent les opérateurs : et : : de SCILAB. En resumé :

<code>n:m</code>	nombres de <code>n</code> à <code>m</code> par pas de 1
<code>n:p:m</code>	idem, par pas de <code>p</code>
<code>length(x)</code>	taille de <code>x</code>
<code>x(i)</code>	ieme coordonnée de <code>x</code>
<code>x(\$)</code>	dernière coordonnée de <code>x</code>
<code>x(i1:i2)</code>	coordonnées <code>i1</code> à <code>i2</code> de <code>x</code>

Essayez les instructions suivantes :

```
v=0:2          // vecteur [0,1,2]
2:0.5:4        // nombres de 2 a 4 par pas de 0.5
5:-1:1         // vecteur [5 4 3 2 1]
v'             // conjugué de v
v'*v, v*v'     //
v($)           // dernier element
v.*v           // multiplication terme a terme.
ones(v)        // vecteur de 1 de meme taille que v
ones(v)+v      // ajouter 1 a toutes les composantes
1+v            // idem (le scalaire 1 est redimensionné comme v)
ones(v)./(1+v) // division element par element
```

Variables prédéfinies et instructions diverses. Les variables prédéfinies commencent en général par un % et sont définies initialement par SCILAB. Voir annexe pour une liste.

```
%i^2, %pi      // %i, %pi, %e .. sont des variables predefinies
exp(%i*%pi)    // Résultat à environ 1e-16 près
1 + 2^3/4      // Eventuellement, pour le caractere ^
               // utilisez la combinaison clavier " Alt Gr + ^ "
3e-6           // 3 * 10^(-6)
j=%i, j, j^2   // Comprendre les résultats obtenus.
1==1, 0~=0     // == : Test d'egalite, ~= : de difference
1<=0           // Les resultats sont les valeurs booleennes
               // %T ou %F (ou: %t ou %f)

x=(1>=0)
y=%T
x==y           // Test d'egalite
```

```

[1 2 3]==[1 2 0]          // test d'egalite, terme a terme

%eps                      // precision machine, de l'ordre de 1e-16
1+%eps == %eps
1+%eps/2 == %eps

format(20)                // Affichage avec 20 caractères.
1.1                      // Qu'en pensez vous ?
format(10)                // Retour affichage normal.

```

Pour voir toutes les variables, on peut utiliser les commandes `whos()` ou `who` (éventuellement après avoir redimensionné la fenetre SCILAB).

Il est important de savoir que SCILAB travaille en précision finie. C'est pourquoi le test $1+\%eps/2==1$ est vrai en SCILAB. De même, on trouverait que $M + \delta = M$ pour toute constante M et tout δ t.q. $|\delta| \leq \frac{\epsilon}{2}|M|$. Il faut aussi comprendre que `%\eps` est une erreur relative qui peut être générée à chaque calcul.

Extraction de sous matrices.

```

a=[1 2; 3 4; 5 6]
a(2:3,1:2)          // Extraction des lignes 2 a 3 et des colonnes 1 a 2.
a(1,:)              // b = ligne 1 de a   (ici : = tous les indices colonnes)
a(:,2)              // colonne 2 de a     (ici : = tous les indices lignes)
a($,1)              // $ = le dernier element

b=a(:)              // vecteur constitue' des colonnes de a
matrix(b,3,2)       // reconstruire la matrice (3,2) correspondante

```

Encore quelques manipulations

```

a=[1 2 3 ; 4 5 6 ; 7 8 9];
2*a, a/4
a^2
a.^2                // Elevation au carre element par element
a.*a                // multiplication element par element

```

Exemple : Creation d'une matrice bande.

On desirer créer une matrice tridiagonale de taille $n \geq 1$ variable, avec $a_{i,i} = 2$ et $a_{i,i+1} = a_{i+1,i} = -1$ pour tout i . Tester et comprendre les 3 solutions suivantes :

Solution 1. Avec boucle `for/end` :

```

n=5;
A=zeros(n,n);
for i=1:n, A(i,i)=2; end
for i=1:n-1, A(i,i+1)=-1; A(i+1,i)=-1; end

```

Solution 2.

```
n=5;
A = 2*diag(ones(n,1))-diag(ones(n-1,1),-1)-diag(ones(n-1,1),1)
```

Solution 3.

```
n=5;
J=diag(ones(n-1,1),1);
A= 2*eye(J) - J - J'
```

Resoudre un systeme linéaire : le solveur \ (ou "backslash") de Scilab.

Supposons maintenant qu'avec l'exemple précédent on veuille résoudre le système linéaire $Ax = b$ avec $b = (1, \dots, 1)^T$. La commande `A\b` renvoie un vecteur x solution de $A*x=b$. On peut procéder comme suit :

```
b=ones(n,1)
x=A\b
norm(A*x-b) // pour verification !
```

On aurait pu aussi utiliser l'inverse de A :

```
x=inv(A)*b
```

Si possible, il vaut mieux éviter de calculer l'inverse de la matrice (avec `inv(A)` par exemple) car cela est beaucoup plus coûteux (et moins précis) que la résolution d'un système linéaire.

D'autres méthodes de résolution de systèmes linéaires sont possibles : méthode LU (c.f. `lu`, `lusolve`), méthodes itératives, ...

3 Programmer en SCILAB

3.1 Commande EXEC

Nous allons maintenant programmer en SCILAB. La commande `exec`, sous SCILAB, va permettre d'exécuter un fichier d'instructions SCILAB.

Pour éditer un fichier, le plus simple est d'utiliser l'éditeur intégré de SCILAB ("Scipad"). Cliquez sur "Editor" en haut à droite de la fenêtre SCILAB. On peut lancer des exécutions de script aussi à partir de cet éditeur (Cliquez sur `Execute > Load into Scilab`) On peut aussi utiliser l'éditeur de son choix, par exemple `kedit`, `xemacs`, ...¹

Mettez quelques instructions SCILAB dans ce fichier. Par exemple

```
a=1; b=2
c=3
```

Sauvegardez votre fichier (sous le nom `toto.sce`) Puis exécutez dans SCILAB la commande suivante :

¹Dans ce cas, par exemple dans la fenêtre "XTERM", tapez `"KEDIT TOTO.SCE &"` ce qui va permettre d'écrire dans un fichier de nom "TOTO.SCE" (après avoir sauvegardé ce fichier).

```
exec toto.sce
```

Ceci execute toutes les instructions qui se trouvent dans le fichier `toto.sce`.

On peut faire de manière équivalente : `exec('toto.sce')`

On peut aussi executer `toto.sce` avec la souris dans SCILAB : > File > File Operations > puis sélectionner le fichier `toto.sce` > Exec.

Remarque. La convention est d'utiliser les extensions `*.sce` pour des fichiers executables, et `*.sci` pour des fichiers de fonctions.

3.2 Organisation du bureau et conseils

Dans la suite, on pourra travailler sur *un seul écran* avec :

- la fenêtre SCILAB,
- l'éditeur de fichier,
- + éventuellement une fenêtre graphique SCILAB

c'est à dire avec toutes ces fenêtres visibles *simultanément*.

Choisir la taille des fontes : En principe, en cliquant sur la touche `Ctrl` (touche "contrôle"), maintenue enfoncée, puis sur le bouton droit de la souris (la souris étant dans la fenêtre SCILAB), vous pouvez choisir la taille des fontes (par exemple `small`, ou `large...`).

Ajuster la dimension de la fenêtre SCILAB : à l'aide de la souris, prendre un coin de la fenêtre SCILAB avec le bouton gauche, et ajuster.

Copier-Coller avec la souris sous linux. Vérifiez que vous arrivez à faire des opérations du type copier-coller avec la souris. ²

3.3 Commande `MODE`

Maintenant, modifiez votre fichier de la façon suivante (puis executer) :

```
// FICHIER toto.sce
mode(-1)
a=1; b=2
c=3
```

Le resultat est que Scilab n'affiche plus rien apres `mode(-1)`, bien qu'il execute toutes les commandes. On le vérifie en regardant les valeurs de `a`, `b`, `c`. Pour voir un résultat quelconque, on peut forcer l'affichage avec les commandes `disp` ou `printf` de SCILAB. Exemple :

```
// FICHIER toto.sce
mode(-1)
a=1; b=2
```

²Par exemple avec la souris sélectionner les instructions dans la fenetre "éditeur de fichier" (click souris bouton gauche pour selectionner la fenetre ; click bouton gauche maintenu pour sélectionner la zone, déclicker une fois la zone sélectionnée ; l'opération "copier" est terminée) ; Puis allez dans la fenêtre SCILAB (click gauche dans la fenêtre SCILAB) ; Puis "coller" à l'aide d'un click bouton milieu. Si le bouton milieu ne marche pas, il est en général simulé par l'appui simultané des boutons droit et gauche de la souris... NB : Il est aussi possible que cela ne marche pas sur certaines machines mal configurées.

```

c=3
disp([a,b]);           // afficher [a,b]
printf("a=%f, b=%5.2f",a,b); // affichage avec format spécifique (f=float)

```

Voici quelques modes utiles.

```

mode(-1) : pas d'affichage (sauf graphes, "disp" ou "printf").
mode(0) : affiche toutes les réponses.
mode(1) : affiche instructions + réponses (mode par défaut).
mode(7) : attend un Enter après chaque instruction.

```

On peut aussi décider directement du mode d'exécution. Par exemple pour exécuter en mode -1 :

```

exec('toto.sce',-1)

```

3.4 Arrêt en cours d'exécution et "débuggage"

- Afin d'arrêter un programme en cours d'exécution, on peut taper au clavier **^C** (Appui sur la touche clavier **Ctrl**, et en maintenant enfoncé, appui sur la touche **C**). Si cela ne marche pas, on peut aussi utiliser la souris : **> Control > stop** (Bouton "Control" puis bouton "stop" dans la fenêtre SCILAB). On peut enfin "programmer" un arrêt dans un exécutable Scilab, avec la commande **pause**.

On rentre alors en mode "débuggage", le prompt suivant s'affiche dans la fenêtre :

```

-1->

```

(Si on réitère l'opération, des modes de débbugage suffessif se succèdent : -2->, etc.)

- Pour revenir au mode normal, on peut utiliser soit la commande

```

-1-> resume

```

```

-->

```

qui a pour effet de reprendre l'exécution du programme après un arrêt, soit la commande **abort**, qui a pour effet de d'arrêter l'exécution du programme et de se replacer en mode normal d'exécution. Exemple :

```

--> pause

```

```

-1-> abort

```

```

-->

```

Cette commande peut aussi s'utiliser dans un script SCILAB afin de forcer l'arrêt définitif du programme :

```

for i=0:10;
    disp(i)
    if i==5; abort; end
end

```

On aurait de même utilisé la commande **pause** (au lieu de **abort**) afin de passer en mode débogage lorsque **i=5**. En Résumé :

^C, pause	arrêt du programme, mise en mode debuggage -1->
resume, return	reprise de l'exécution du programme
abort	arret définitif du programme, retour au mode normal

4 Affichage

Commande disp. Cette commande permet d'afficher n'importe quelle variable, matrice ou chaîne de caractère, sans se soucier du format d'affichage. Exemple :

```
A=rand(2,2);
disp(A);
```

Commande printf. Cette commande permet d'afficher en spécifiant un format. La syntaxe est du type **printf('chaîne de caractère',x1,x2,...)** Dans la chaîne de caractère, les variables **x1, x2,...** sont affichées successivement suivant le format spécifié dans la chaîne de caractère et commençant par **%** pour chaque nouvelle variable (et **\n** pour un passage à la ligne). Exemple :

```
a=1/3; b=200; printf('a=%f, b=%i',a,b);
```

Ici **%f** spécifie un format "float" c'est à dire le format double précision (ou 16 décimales). Les formats les plus courants sont :

%f	Float
%e	Float, format $a \times 10^b$
%i	entier
%s	string (chaîne de caractères)
\n	passage à la ligne

Autres exemples :

```
a=1/3; b=1; printf('a=%10.5f\nb=%5i',a,b);
x=(1:3)'; y=ones(x); printf('x=%i, y=%i\n',x,y);
```

Dans le premier exemple, **%10.5f** est un format **float** avec 10 caractères en tout, dont 5 après la virgule ; **%5i** est un format d'entier avec 5 caractères. Dans le deuxième exemple, on voit que **printf** peut gérer l'affichage de vecteurs colonnes.

Commande scanf. Cette commande permet de rentrer des variables avec un format prédéfini. Elle s'utilise de manière analogue à **printf**. Exemples :

```
printf('entrer un reel:');
x=scanf('%f')
```

Pour attendre l'appui sur "Return" dans un programme, on pourra utiliser simplement

```
scanf('')
```

5 Boucles et Tests

5.1 Tests

Les instructions principales sont `if-(else)-end` et `select-case-(else)-end`. On pourra essayer tous les tests qui suivent en mettant les instructions dans un fichier puis en exécutant sous SCILAB.

L'instruction **if-(else)-end** :

```
a=0;
if a==0
    disp('a nul')
end
```

On peut aussi traiter les "autres cas" avec la commande `else` :

```
a=1;
if a>0
    disp('a positif ')
else
    disp('a negatif ou nul')
end
```

L'instruction **select-case-(else)-end**. Permet de sélectionner un traitement suivant la valeur choisie :

```
select a
case 0; disp('a=0');
case 1; disp('a=1')
else disp('a different de 0 et 1')
end
```

5.2 Boucles

Voici 2 manières équivalentes de programmer des boucles. On donne dans chaque cas un exemple d'arrêt si `i>5` est "vrai" (arrêt=sortie de la boucle)

Avec la commandes for-end :

```
// boucle simple
for i=0:5
    i,
end;

// boucle avec arrêt
for i=0:10
    if i>5; break, end    // break permet de sortir de la boucle.
    disp(i);
end;
```

Avec la commandes while-end :

```
i=0;
while (i<=5)
    disp(i)
    i=i+1;
end
```

Noter qu'on peut rentrer une liste d'indices non equirépartis :

```
for a=[1 2 4 2.1]
    printf('a=%e\n',a)
end
```

5.3 Commande find

Voici une commande tres utile pour eviter de faire des boucles, dans le cas par exemple où l'on cherche les éléments d'un vecteur ou d'une matrice vérifiant certaines propriétés. De facon generale la commande `find(X)` permet de trouver tous les indices `i` tels que `X(i)` soit vrai.

Par exemple la commande `find(x>a)` permet de trouver tous les indices `i` t.q. `x(i)>a`, en supposant que `x` est un vecteur et `a` une constante :

```
a=1;
x=(0:0.1:2)';
i=find(x>a) // liste ligne
x(i)
```

De meme pour trouver des elements `i` t.q. $x_i \in]a, b]$ par exemple :

```
x=rand(10,1)
a=0.2; b=0.8;
i=find(a<x & x<=b)
x(i)
```

Remarque : La commande `max` (resp. `min`), qui renvoie le maximum (resp. le minimum) d'un vecteur ou d'une matrice, fonctionne de maniere similaire à `find` pour la gestion des indices.

Complément (en seconde lecture). On a de même pour `X` matrice :

```
X=rand(3,3)
i=find(X>0.5)
X(i)
```

Ici les elements de la matrices sont ranges suivant un certain ordre : la matrice $X = [X_1, \dots, X_P]$

(liste des vecteurs colonnes) est réordonnée en un seul vecteur $\begin{pmatrix} X_1 \\ \vdots \\ X_P \end{pmatrix}$ avant la detection d'indices.

On peut avoir les veritables indices `i` et `j` si on les demande explicitement :

```

X=rand(3,3)
[i,j]=find(X>0.5)
// affichage des valeurs X(i,j):
i=i'; j=j'; xval=X(i+3*(j-1));
printf('X(%i,%i)=%f\n',i,j,xval)

```

6 Définition de fonctions

• Pour définir la fonction $f(x) = x + 1$ on peut procéder comme suit, directement dans SCILAB :

```
function y=f(x); y=x+1; endfunction
```

C'est ce qu'on appellera la **définition en ligne**. Puis toujours dans SCILAB, essayez

```

f(0)
f(1)

```

Dans la définition, x est une variable d'entrée (ou "input"), y est une variable de sortie (ou output). La partie $y=f(x)$ donne la structure de la fonction. La partie interne, $y=x+1$, définit la variable de sortie en fonction de la variable d'entrée. La syntaxe générale est `function [y1,...,yn]=f(x1,...,xp)`. (faire `help function` pour plus de détails).

• On peut aussi programmer des fonctions dans un fichier exécutable SCILAB. Par exemple modifions le fichier `toto.sce` comme suit :

```

// fichier toto.sce
function y=f(x)
    y=x+1
endfunction
function y=f2(x); y=x^2; endfunction;
for i=0:10
    printf('%2i %10.2f %10.2f\n',[i,f(i),f2(i)]);
end

```

Puis on exécute avec `exec toto.sce` dans SCILAB. On peut donc mélanger des définitions de fonctions **et** des instructions SCILAB classiques.

• Lorsque les fichiers deviennent important, il est préférable de mettre les définitions de fonction dans un fichier séparé et utiliser `getf`. Par exemple, créer le fichier `func.sci` contenant :

```

// fichier func.sci
function y=f(x)
    y=x+1
endfunction

```

Puis faire dans SCILAB :

```
getf func.sci
f(2)
f(3)
```

La commande `getf` permet donc de "charger" un fichier de fonctions dans la fenêtre SCILAB. Dans ce cas, le fichier `func.sci` ne doit contenir **que** des définitions de fonctions.

Exercice. 1 Définir (en ligne) une fonction de nom `alea` qui a un entier `n` renvoie une matrice carrée aléatoire de taille `n`.

Exercice. 2 Définir dans un fichier `func.sci` une fonction de nom `MAT` qui a un entier `n` renvoie une matrice carrée tridiagonale t.q. $a_{i,i} = 2$, $a_{i,i+1} = a_{i+1,i} = -1 \forall i$. Tester votre programme avec la commande `exec` et diverses valeurs de `n`.

Solutions

```
//SOLUTION EXO 1
function A=alea(n); A=rand(n,n); endfunction;
```

Pour l'exercice 2, dans `func.sci` :

```
// SOLUTION EXO 2 - Premiere solution:
function A=MAT(n)
  A=zeros(n,n);
  for i=1:n; A(i,i)=2; end;
  for i=1:n-1; A(i,i+1)=-1; A(i+1,i)=-1; end;
endfunction
MAT(3)
```

puis dans SCILAB faire `exec func.sci`

Deuxieme solution, sans boucle for/end :

```
function A=MAT(n)
  w=ones(n-1,1);
  A=2*eye(n,n) - diag(w,1) - diag(w,-1);
endfunction
MAT(3)
MAT(10)
```

7 Graphiques

7.1 Graphiques 2d

Le graphisme en Scilab est relativement simple à utiliser.

Pour des graphes élémentaires on peut utiliser la commande `plot`. Si `x,y` sont des vecteurs de même longueur, `plot(x,y)` va faire un graphe continu affine par morceaux passant par les points $(x(i), y(i))$. On pourra faire `help plot` pour avoir un descriptif général, et `help Graphics` pour accéder à l'ensemble des commandes graphiques, (et aussi `help xset` pour accéder à des réglages particuliers).

Exemple. Le graphe de $y = x^2$ peut être obtenu comme suit :

```
x=(-1:0.1:1)';
y=x^2;
plot(x,y);
```

Il est important que `x` et `y` aient la même longueur. En cas de problème, on peut vérifier les longueurs avec les commandes `length(x)` ou `size(x)`.

On peut aussi rajouter un titre et un texte (ou label) en abscisse et en ordonnée, avec la commande `xtitle`. Par exemple :

```
xtitle('quadrique','x','x^2')
```

La commande `clf` (ou `xbasc`), initialise la fenêtre graphique courante, ou bien efface la fenêtre courante si elle existe déjà.

```
clf;
```

On peut mettre une couleur avec un paramètre optionnel

```
clf;
plot(x,sin(3*x),'r');      // 'r' ou 'red'
plot(x,sin(2*x),'grey--'); // 'g' ou 'grey' , style lignes brisées
```

Pour voir les couleurs et styles prédéfinies, voir `LineSpec` :

```
help LineSpec
```

Exemple. Tracer les deux courbes x^2 et x^3 sur $[-1, 1]$ (tester avec la commande `exec`) :

```
x=(-1:0.1:1)';
clf;
plot(x,x^2,x,x^3);
scanf('%c')

// Et on peut spécifier divers styles:
clf
plot(x,x^2,'-',x,x^3,'r--');
```

7.2 Plusieurs graphiques (en seconde lecture)

- On peut mettre plusieurs graphiques sur une même fenêtre graphique avec la commande `subplot` (ou `xsetech`) Exemple :

```
x=(0:0.1:1)';
clf
subplot(211) // 2 divisions en y, 1 division en x, trace' dans 1iere zone
plot2d(x,x^2)
subplot(212) // 2 divisions en y, 1 division en x, trace' dans 2ieme zone
plot2d(x,sin(x))
```

- On peut aussi travailler avec plusieurs fenêtres graphiques en utilisant `xset('window',n)` où n est le numéro de la fenêtre graphique. Exemple :

```

x=(0:0.1:1)';
xset('window',0)
clf
plot(x,x^2)
xset('window',1)
clf
plot(x,sin(x))

```

7.3 Légendes, dimensions de la fenêtre (en seconde lecture)

On peut afficher une légende générale par la commande `legend` (`help legend` pour plus de détails).

Voici un exemple pour le tracé de $\sin(x)$ et $\sin(2x)$ avec une légende en haut à gauche (résultat figure 1). On a de plus imposé une fenêtre de taille $[-1, 1] \times [-0.5, 0.5]$:

```

x=(-1:0.1:1)'; y1=sin(x); y2=sin(2*x);
clf
plot(x,y1,'b-');
plot(x,y2,'r--');
legend(['titre1','titre2'],2);    // 2: paramètre optionel pour situer la légende

a=gca();
a.data_bounds=[-1,1,-0.5,0.5];    // du type [Xmin,Xmax,Ymin,Ymax]

```

Voir `axis_properties` pour le contrôle d'autres paramètres graphiques.

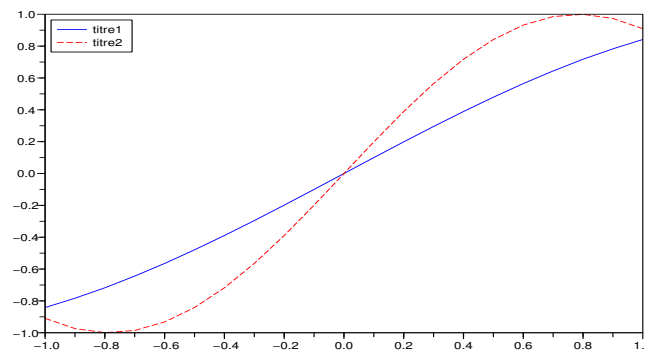


FIG. 1 – Utilisation de la commande LEGENDS

Remarque 1 Il existe aussi d'autres méthodes pour obtenir les graphiques 2d. Par exemple avec la commande `plot2d`, pour fixer les dimensions de la fenêtre graphique :

```

clf
plot2d(x,x^2,rect=[0,0,1,0.8])

```

(syntaxe : `rect=[Xmin,Ymin,Xmax,Ymax]`) Pour préciser la couleur :

```

plot2d(x,x^2,style=2,rect=[0,0,1,0.8])

```

Ici `style=2` représente la couleur "bleue" ; faire `xset()` pour voir les numéros des couleurs prédéfinies.

7.4 Graphiques 3d

Il s'utilise de manière similaire. Voici un exemple pré-programmé :

```
plot3d()
```

8 Fin de session

1. Quittez SCILAB (souris : "File/Quit", ou tapez "quit" dans SCILAB)
2. Eventuellement, verifier que le "processus" SCILAB (**scilex**) ne continue pas de fonctionner. Pour cela, faites sous linux (fenetre **Xterm**) la commande

```
ps
```

pour verifier qu'il n'y a plus de processus **scilex** à votre nom qui tourne. Le cas échéant, tuez le processus "scilex", en faisant

```
killall scilex
```

3. Quittez votre session linux.

9 Obtenir SCILAB

Voir <http://www.scilab.org>

Résumé de quelques commandes SCILAB

Constantes prédéfinies	
%pi	3.1415927..
%e	2.7182818..
%i	$\sqrt{-1}$
%eps	2.22E-16
%inf	+infini
%t,%T	vrai
%f,%F	faux
%s	variable polynôme

Vecteurs	
n:m	nombres de n à m par pas de 1
n:p:m	idem, par pas de p
length(x)	taille de x
x(i)	ième coordonnée de x
x(\$)	dernière coordonnée de x
x(i1:i2)	coordonnées i1 à i2 de x
x'	vecteur transposé
x'*y	produit scalaire des vecteurs (colonnes) x et y
x*y'	matrice $(x_i y_j)$ si les vecteurs x et y sont colonnes
Note : En SCILAB, les vecteurs sont en fait des matrices lignes ou colonnes.	

Matrices	
<code>size(A)</code>	nombres de lignes et colonnes de A
<code>size(A,'r'), size(A,'c')</code>	nombre de lignes, nombre de colonnes
<code>A(i,j)</code>	A_{ij}
<code>A(i,:)</code>	i-ième ligne de A
<code>A(:,j)</code>	j-ième colonne de A
<code>A(:,j1:j2)</code>	colonnes j1 à j2 de A
<code>A(\$,:)</code>	dernière ligne de A
<code>A(:, \$)</code>	dernière colonne de A
<code>C=[A,B], C=[A B]</code>	concaténation $C=[A \ B]$
<code>C=[A;B]</code>	concaténation $C = \begin{pmatrix} A \\ B \end{pmatrix}$
<code>A'</code>	Matrice conjuguée (= matrice transposée dans le cas réel)
<code>A.'</code>	Matrice transposée
<code>zeros(m,n)</code>	matrice nulle de taille (m,n)
<code>ones(m,n)</code>	matrice de taille (m,n) , coefficients égaux à 1
<code>eye(m,n)</code>	matrice de taille (m,n) , coefficients diagonaux égaux à 1
<code>eye(A)</code>	idem, avec les dimensions m,n de la matrice A
<code>diag(A)</code>	vecteur contenant les éléments diagonaux de A
<code>diag(x)</code>	Matrice diagonale des éléments du vecteur x
<code>triu(A)</code>	Matrice triangulaire supérieure des éléments de A
<code>tril(A)</code>	Matrice triangulaire inférieure des éléments de A
<code>rand(m,n)</code>	matrice aléatoire de taille m,n (loi uniforme $U(0,1)$)
<code>rand(m,n,'normal')</code>	matrice aléatoire de taille m,n (loi normale $N(0,1)$)