

Scilab

I. Premier pas

1. Qu'est-ce que Scilab?

- Scilab est un logiciel de calcul matriciel du type Matlab (contrairement à Maple qui est un programme de calcul formel et symbolique). Les objets mis en oeuvre dans Scilab sont des matrices (même les scalaires qui sont considérés comme des matrices 1×1). Les résultats renvoyés sont numériques.

- Contrairement à Matlab, Scilab est gratuit d'utilisation. Pour le reste, ce sont des logiciels très similaires dans l'utilisation. En particulier, Scilab intègre des fonctions Matlab.

2. Première utilisation de Scilab

Les instructions de Scilab sont tapées dans la fenêtre de commande du logiciel appelée console.

Entrer les commandes après les sigles `-->`, valider en tapant sur la touche `entrée` et vérifier le résultat.

```
--> 2+1
ans =
    3.
--> 2*%pi
ans =
    6.2831853
--> %eps
%eps =
    2.220D-16
--> 2*%eps
ans =
    4.441D-16
--> 2+%eps
ans =
    2.
--> %i
%i =
    i
--> 2+3*%i
ans =
    2. + 3.i
--> x=5*7
x =
    35.
--> x
x =
    35.
--> y=exp(x)
y =
    1.586D+15
```

```

--> 1+2+3+4+5...
--> +6+7
ans =
    28.
--> rand('uniform')
--> format('v',10)
--> z=rand()
z =
    0.8782165
--> format(20)
--> z
z =
    0.87821648130193353
--> format('e',10)
--> z
z =
    8.782D-01
--> format(20)
--> z
z =
    8.7821648130193D-01
--> u=3;v=4,w=5;
v =
    4.
--> typeof(v)
ans =
    constant
--> whos
Name                Type                Size                Bytes
whos                function                9000
w                   constant                1 by 1                24
v                   constant                1 by 1                24
u                   constant                1 by 1                24
z                   constant                1 by 1                24
y                   constant                1 by 1                24
x                   constant                1 by 1                24
home                string                  1 by 1                160
...
--> clear
--> whos
Name                Type                Size                Bytes
whos                function                9000
home                string                  1 by 1                160
...
--> exit

```

- L'affectation d'un calcul à une variable se fait via l'opérateur =.
- Le résultat d'une affectation est affiché à l'écran sauf si cette affectation est suivie du sigle ;.
- On peut effectuer plusieurs opérations sur une même ligne en les séparant par le sigle , si l'on désire un affichage et par le sigle ; dans le cas contraire.
- Le nom d'une variable ne doit pas être choisie parmi les mots réservés du langage.
- Par défaut, tout calcul est affecté à la variable ans.
- Les variables %pi, %eps, %i, sont déjà affectées.
- La commande format modifie l'affichage du format des différentes variables.
- Les commandes who, whos listent l'ensemble des variables en cours.
- La commande clear efface le contenu de toutes les variables utilisées.

- La commande `. . .` en fin de ligne permet de continuer la même expression sur une ligne suivante (la touche `entrée` n'est alors pas reconnue comme une validation de la ligne).
- Lorsque l'on veut interrompre un programme, la commande `CTRL-C` permet de récupérer la main.
- La commande `help` ouvre une fenêtre d'aide.
- La commande `help` suivi du nom d'une commande affiche l'aide sur la commande.
- Il n'est pas possible de sauvegarder la fenêtre de commande de **Scilab**.
Pour de nombreux exercices, il sera préférable d'utiliser un éditeur de texte (peu importe lequel) pour enregistrer les commandes.
Les fichiers de commandes doivent avoir l'extension `.sci`
Après avoir défini le répertoire de travail, tout fichier à l'extension `.sci` peut être chargé par **Scilab** en utilisant la commande `getf`.
- Un certain nombre de commandes **Matlab** sont utilisables dans **Scilab**. Ces commandes sont facilement reconnaissables car elles sont préfixées par `"mtlb_"`. Par exemple, `mtlb_format`, `mtlb_axis`, `mtlb_max` ou `mtlb_sort`.

Il existe des fonctions spécifiques aux complexes :

```
abs(z)    module du complexe z
real(z)   partie réelle du complexe z
imag(z)   partie imaginaire du complexe z
sqrt(z)   racine carrée du complexe z
```

```
--> z=2*%i
z =
    2.i
--> real(z)
ans =
    0.
--> abs(z)
ans =
    2.
--> sqrt(z)
ans =
    1. + i
```

II. Création de matrices

Une matrice peut être définie élément par élément. Pour cela, les valeurs sont entre crochets, le séparateur de colonnes est l'espace ou la virgule. Celui de lignes est le point-virgule.

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

Il existe des matrices prédéfinies.

`zeros(n,m)` construit une matrice de taille $n \times m$ ne contenant que des 0.

`ones(n,m)` construit une matrice de taille $n \times m$ ne contenant que des 1.

`eye(n,m)` construit une matrice de taille $n \times m$ dont les éléments diagonaux sont 1 les autres étant nuls.

`rand(n,m)` construit une matrice de taille $n \times m$ dont tous les éléments sont choisis aléatoirement dans l'intervalle $[0; 1]$ selon une loi uniforme.

```

--> B=zeros(2,3)
B =
    0.    0.    0.
    0.    0.    0.
--> C=ones(2,1)
C =
    1.
    1.
--> Id=eye(3,2)
Id =
    1.    0.
    0.    1.
    0.    0.
--> D=rand(3,3)
D =
    0.2113249    0.3303271    0.8497452
    0.7560439    0.6653811    0.6857310
    0.0002211    0.6283918    0.8782165

```

On peut accéder à un élément d'une matrice, l'utiliser ou en modifier sa valeur en spécifiant ses références ligne et colonne.

```

--> A=rand(4,4)
A =
    0.0683740    0.1985144    0.2164633    0.9329616
    0.5608486    0.5442573    0.8833888    0.2146008
    0.6623569    0.2320748    0.6525135    0.312642
    0.7263507    0.2312237    0.3076091    0.3616361
--> A(1,2)
ans =
    0.1985144
--> X=A(1,1)
X =
    0.0683740
--> A(1,2)=0.01
A =
    0.0683740    0.01    0.2164633    0.9329616
    0.5608486    0.5442573    0.8833888    0.2146008
    0.6623569    0.2320748    0.6525135    0.312642
    0.7263507    0.2312237    0.3076091    0.3616361

```

On peut aussi définir les éléments de la matrice un par un et en ajouter si nécessaire. Pour cela, on définit directement l'élément en précisant sa position (ligne, colonne).

Lorsque l'élément ajouté est en dehors de la taille originale de la matrice, celle-ci est automatiquement adaptée et les éléments supplémentaires non définis prennent la valeur 0.

```

--> F=[1 2 3 4;5 6 7 8;9 10 11 12; 13 14 15 16]
F =
    1.    2.    3.    4.
    5.    6.    7.    8.
    9.   10.   11.   12.
   13.   14.   15.   16.
--> F(1,5)=17
F =
    1.    2.    3.    4.   17.
    5.    6.    7.    8.    0.
    9.   10.   11.   12.    0.
   13.   14.   15.   16.    0.

```

```
--> F(4,3)=22
F =
  1.    2.    3.    4.   17.
  5.    6.    7.    8.    0.
  9.   10.   11.   12.    0.
 13.   14.   22.   16.    0.
```

La commande `size` permet d'obtenir la dimension de la matrice. Plusieurs options d'utilisation sont possibles (voir l'aide : taper la commande `help size`). On peut aussi utiliser la commande `length` quand on désire le nombre total d'éléments de la matrice.

```
--> size(F)
ans =
  4.    5.
--> size(F,1)
ans =
  4.
--> size(F,2)
ans =
  5.
--> [n,p]=size(F)
p =
  5.
n =
  4.
--> length(F)
ans =
  20
```

Les éléments d'une matrice reçoivent dans **Scilab** une numérotation colonne après colonne. On peut donc accéder à un élément d'une matrice en spécifiant son numéro d'ordre et non pas un couple de référence ligne et colonne.

```
--> F=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12]
F =
  1.    2.    3.    4.
  5.    6.    7.    8.
  9.   10.   11.   12.
--> F(7)
ans =
  3.
```

Dans le cas d'une matrice colonne ou d'une matrice ligne à plus de 2 éléments, il est possible d'ajouter des éléments en spécifiant leur indice dans la numérotation colonne après colonne. Comme pour une numérotation ligne-colonne, si nécessaire, des 0 sont ajoutés.

```
--> clear
--> U=[2 4]
U =
  2.    4.
--> U(3)=6
U =
  2.    4.    6.
--> U(5)=10
U =
  2.    4.    6.    0.   10.
```

```

--> V=[2;4]
V =
    2.
    4.
--> V(4)=8
V =
    2.
    4.
    0.
    8.
--> W(1)=3
W =
    3.
--> W(2)=4
W =
    3.
    4.
--> size(U)
ans =
    1.    5.
--> size(V)
ans =
    4.    1.
--> size(W)
ans =
    2.    1.
--> length(V)
ans =
    4.

```

Attention, la même commande pour un autre type de matrice crée une erreur :

```

--> F=[1 2 3 4 ; 5 6 7 8 ; 9 10 11 12];
--> F(15)=3
      !--error 21
Index invalide.

```

On peut définir directement des intervalles de valeurs et les affecter à un vecteur. La commande `min:pas:max` est l'ensemble des éléments de la forme `min + pas * k` inférieurs à `max` avec `k` un entier naturel. Si la valeur de `pas` n'est pas défini, il s'agit de 1 par défaut.

```

--> 1:7
ans =
    1.    2.    3.    4.    5.    6.    7.
--> 1:0.1:2.9
ans =
      column 1 to 10
    1.    1.1    1.2    1.3    1.4    1.5    1.6    1.7    1.8    1.9
      column 11 to 20
    2.    2.1    2.2    2.3    2.4    2.5    2.6    2.7    2.8    2.9
--> X=[1:3]
X =
    1.    2.    3.

```

La commande `linspace(deb,fin,n)` donne un résultat similaire mais pas nécessairement identique. Elle crée un vecteur ligne de `n` points à intervalles réguliers entre `deb` et `fin`. Les valeurs `deb` et `fin` sont la première et la dernière du vecteur ligne. Ce qui n'est pas nécessairement le cas pour le commande `min:pas:max`.

```
--> U=linspace(0,5,10)
U =
    column 1 to 5
    0.    0.5555556    1.1111111    1.6666667    2.2222222
    column 6 to 10
    2.7777778    3.3333333    3.8888889    4.4444444    5.
--> V=linspace(0,5,11)
V =
    0.    0.5    1.    1.5    2.    2.5    3.    3.5    4.    4.5    5.
```

Il est souvent nécessaire de travailler sur les lignes ou les colonnes d'une matrice. Il est possible d'obtenir ces éléments en utilisant éventuellement des intervalles d'éléments. On peut extraire aussi des sous-matrices en spécifiant de façon similaire les intervalles d'indices pour les lignes et les colonnes.

$A(i, :)$ désigne la i ème ligne de la matrice A
 $A(:, j)$ désigne la j ème colonne de la matrice A
 $A(2:4, :)$ désigne la matrice formée des 2ème, 3ème et 4ème lignes de la matrice A.
 $A(:, 1:2:size(A,2))$ désigne la matrice formée des colonnes impaires de la matrice A.

```
--> A=[1 2 3 4;5 6 7 8;9 10 11 12; 13 14 15 16]
A =
    1.    2.    3.    4.
    5.    6.    7.    8.
    9.   10.   11.   12.
   13.   14.   15.   16.
--> A(2,:)
ans =
    5.    6.    7.    8.
--> A(:,3)
ans =
    3.
    7.
   11.
   15.
--> A(2:4, :)
ans =
    5.    6.    7.    8.
    9.   10.   11.   12.
   13.   14.   15.   16.
--> A(:, 1:2:size(A,2))
ans =
    1.    3.
    5.    7.
    9.   11.
   13.   15.
--> A(2:3,2:4)
ans =
    6.    7.    8.
   10.   11.   12.
```

On peut construire une matrice diagonale en donnant simplement les valeurs des éléments diagonaux.

```
--> B=diag([1,2])
B =
    1.    0.
    0.    2.
```

Scilab travaille par défaut avec des matrices. Des fonctions élémentaires comme `exp`, `sin`, `cos`, `abs` agissent donc sur les matrices.

Plus précisément, elles s'appliquent à tous les éléments de la matrice.

```
--> x=[0:0.5:%pi]
x =
    0.    0.5    1.    1.5    2.    2.5    3.
--> sin(x)
ans =
    0.  0.4794255  0.8414710  0.9974950  0.9092974  0.5984721  0.1411200
--> F=[1,2;3,4]
F =
    1.    2.
    3.    4.
--> cos(F)
ans =
    0.5403023  - 0.4161468
    - 0.9899925  - 0.6536436
```

III. Opérations et fonctions sur les matrices

1. Opérations usuelles

L'addition ou le produit de deux matrices et la puissance d'une matrice peuvent s'effectuer en respectant les règles usuelles (mathématiques) ou en effectuant ces opérations termes à termes. Dans ce cas, on ajoute en préfixe un point à l'opération.

```
--> F=[1,2;3,4]
F =
    1.    2.
    3.    4.
--> E=[1,1;1,-1]
E =
    1.    1.
    1.   - 1.
--> F+E
ans =
    2.    3.
    4.    3.
--> F*E
ans =
    3.   - 1.
    7.   - 1.
--> F.*E
ans =
    1.    2.
    3.   - 4.
--> F^2
ans =
    7.    10.
   15.    22.
--> F.^2
ans =
    1.    4.
    9.   16.
```

L'inverse d'une matrice s'obtient simplement en utilisant la commande `inv`.

```
--> inv(F)
ans =
  - 2.      1.
    1.5   - 0.5
```

2. Fonctions usuelles sur les matrices

La transposition, la diagonale, la trace, le rang et le déterminant d'une matrice sont obtenus respectivement par les commandes `'`, `diag`, `trace`, `rank` et `det`.

```
--> F=[1 2 3;4 5 6;7 8 9]
F =
  1.      2.      3.
  4.      5.      6.
  7.      8.      9.
--> F'
ans =
  1.      4.      7.
  2.      5.      8.
  3.      6.      9.
--> det(F)
ans =
  6.661D-16      (euh, normalement c'est 0!?!?)
--> rank(F)
ans =
  2.
--> diag(F)
ans =
  1.
  5.
  9.
--> trace(F)
ans =
  15.
```

3. Autres fonctions

Il existe bon nombre de fonctions applicables aux matrices dans `Scilab`. Il est conseillé de voir pour cela l'aide du logiciel. Notons, néanmoins, quelques fonctions simples :

`flipdim(A,1)` qui inverse l'ordre des lignes dans la matrice A.

`flipdim(A,2)` qui inverse l'ordre des colonnes dans la matrice A.

```
--> B=[1 2 3;4 6 7]
B =
  1.      2.      3.
  4.      6.      7.
--> flipdim(B,1)
ans =
  4.      6.      7.
  1.      2.      3.
--> flipdim(B,2)
ans =
  3.      2.      1.
  7.      6.      4.
```

La commande `rot90` de **Matlab** qui effectue une rotation de la matrice de $\pi/2$ dans le sens trigonométrique s'obtient en faisant la transposée d'un flip des colonnes.

```
--> flipdim(B,2)'  
ans =  
     3.     7.  
     2.     6.  
     1.     4.
```

-----Faire la feuille de TP n°1-----

IV. Graphes

1. Graphe en 2D

Soient U, V et Z trois matrices lignes de taille n .

Pour définir une courbe, on peut utiliser les commande suivantes :

`plot2d(Z)` qui relie les points de coordonnées $(i;Z(i))$

`plot2d(U,V)` qui relie les points de coordonnées $(U(i);V(i))$

```
--> X=[-3*%pi:0.01:3*%pi];  
--> Y=cos(X);  
--> Z=sin(X);  
--> plot2d(Y)  
--> xbascc()  
--> plot2d(X,Z)
```

La commande `xbascc(num)` permet d'effacer la fenêtre numéro `num`.

La commande `close(num)` ferme la fenêtre numéro `num`.

Pour ces deux commandes, si aucune valeur n'est précisée, c'est la fenêtre en cours qui est prise en compte.

Pour fermer toutes les fenêtres graphiques ouvertes, la commande `mtlb_close all` de **Matlab** est assez efficace.

En cas de commandes d'affichage successives, par défaut, celles-ci se complètent dans la même fenêtre.

```
--> xbascc()  
--> plot2d(X,Y)  
--> plot2d(X,Z)
```

Pour mettre les deux courbes $(X(i);Y(i))$ et $(X(i);Z(i))$ sur la même courbe, on peut aussi utiliser la commande **Matlab** :

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

Si on contraire, on désire obtenir les courbes dans des fenêtres graphique différentes, il faut précédemment en spécifier ou en créer une pour chaque nouvelle courbe. Pour manipuler les fenêtres graphiques, on dispose des fonctions suivantes :

- `xset("window",num)` active la fenêtre numéro `num` comme fenêtre courante. Si aucune fenêtre de ce numéro existe, elle est créée.
- `xdel(num)` détruit la fenêtre numéro `num`.
- `xselect()` met en avant la fenêtre graphique courante.

```

--> clear
--> X=[-3*%pi:0.01:3*%pi];
--> Y=cos(X);
--> Z=sin(X);
--> plot2d(X,Y)
--> xset("window",1)
--> plot2d(X,Z)

```

Il existe un code numérique pour le style de représentation des points d'un graphique ainsi que leur couleur. En voici quelques valeurs :

Code	Signification
0	point
-1	+
-2	x
-3	⊕
-4	losange plein
-5	losange
-6	triangle vers le haut
-7	triangle vers le bas
-8	trèfle
-9	cercle
1	noir
2	bleu foncé
3	vert clair
4	bleu clair
5	rouge
6	violet
7	jaune

Pour connaître toutes les couleurs, entrez la commande `getcolor()`.

On peut aussi modifier le style des courbes d'une fenêtre en ajoutant des paramètres à la commande `xset`.

Les paramètres `color` ou `background` prenant tous les deux une valeur numérique positive (suivant le même code que précédemment) permettent de modifier respectivement la couleur des axes et celle du fond.

Le paramètre `line style` permet de modifier le style de représentation des axes suivant les valeurs suivantes :

1	continu
2	grand tiret
3	petit tiret
4	point tiret

Enfin, un paramètre de style sur une courbe peut être affectée à celle-ci sous la forme d'une troisième composante à la commande `plot2d`.

```

--> clear
--> close
--> xset('background',3)

```

```
--> xset('line style',2)
--> xset('color',2)
--> X=[-3*%pi:0.01:3*%pi];
--> Y=cos(X);
--> Z=sin(X);
--> plot2d(X,Y,4)
```

On représente les courbes paramétrées naturellement par la fonction `plot2d` :

```
--> xset("window",1)
--> t=linspace(-%pi,%pi,500);
--> x=sin(2*t);
--> y=cos(3*t);
--> xbas()
--> plot2d(x,y)
```

La mise en forme du graphique peut se faire via les commandes :

`titlepage('texte')` et `xtitle('texte')` ajoutent le titre `texte` à une fenêtre graphique.
`legend([NomFct1,NomFct2,...],[ColorFct1,ColorFct2,...],[PosX,PosY])`
permet de nommer les courbes du graphe.

`xlabel('legende')` ajoute une légende à l'axe horizontale du graphe.

`ylabel('legende')` ajoute une légende à l'axe vertical du graphe.

```
--> xtitle('sinus et cosinus')
--> legends(["sin","cos"],[5,6],[0,0])
--> xlabel('Axe des abscisses')
--> ylabel('Axe des ordonnées')
```

Pour plus d'informations sur les précédentes commandes, utiliser l'aide de **Scilab**.

La fonction `mtlb_axis` de **Matlab** permet aussi le paramétrage des axes. C'est le cas par exemple lorsque l'on souhaite un repère orthonormal (`mtlb_axis('equal')`).

De façon générale, une méthode simple pour modifier les paramètres d'une courbe consiste à le faire via le menu de la fenêtre graphique (Edit->Propriété de la figure).

2. Graphe en 3D

La commande `plot3d3(X,Y,Z)` trace le graphe des points $(X(i);Y(i);Z(i))$ si X, Y et Z sont trois vecteurs de même taille n .

```
--> clear
--> mtlb_close all
--> X=[-2*%pi:0.01:2*%pi];
--> Y=cos(X);
--> Z=sin(X);
--> plot3d3(X,Y,Z)
```

Une figure identique peut être obtenue en utilisant la commande pour les courbes paramétrées en 3D :

```
--> param3d(X,Y,Z)
```

Si l'on désire le graphe d'une surface c'est-à-dire un ensemble de points de la forme $(i,j,f(i,j))$, on utilise les commandes `plot3d` ou `plot3d1`.

```
--> clear
--> mtlb_close all
--> i=[-2:0.05:2];
--> j=[0:0.05:6];
--> T=cos(i'*j);
--> plot3d(i,j,T)
```

On peut obtenir des effets de couleurs :

```
--> mtlb_close all
--> xset('colormap',hotcolormap(128))
--> plot3d1(i,j,T)
```

La modification des axes et l'ajout d'annotations graphiques sont les mêmes que pour les graphes en 2D.

Utilisez la commande `help plot3d` pour plus d'informations.

3. Commande pour plusieurs figures

On peut créer plusieurs graphes sur une même figure.

Pour cela il faut savoir que les points de la fenêtre graphique sont repérés par leur abscisses et leur coordonnées dans cette fenêtre. Ces coordonnées sont des réels de l'intervalle $[0;1]$. Le coin supérieur gauche a pour coordonnées $(0,0)$ et le coin inférieur droit $(1,1)$.

Toute sous-fenêtre (c'est un rectangle) de la fenêtre graphique peut être ainsi repérée par son coin supérieur gauche et son coin inférieur droit.

Pour sélectionner cette sous-fenêtre comme emplacement où l'on désire une représentation, il suffit d'utiliser la commande `xsetech([x1 y1 dx dy])` où $(x1, y1)$ et $(x1+dx, y1+dy)$ sont les coordonnées respectives du coin supérieur gauche et du coin inférieur droit.

```
--> clear
--> mtlb_close all
--> I=[-%pi:0.01:%pi];
--> J=[-1:0.01:1];
--> X=cos(I);
--> Y=sin(I);
--> U=acos(J);
--> V=asin(J);
--> xsetech([0 0 0.5 0.5])
--> mtlb_axis('equal')
--> plot2d(I,X)
--> xtitle('cosinus')
--> xsetech([0 0.5 0.5 0.5])
--> mtlb_axis('equal')
--> plot2d(I,Y)
--> xtitle('sinus')
--> xsetech([0.5 0 0.5 0.5])
--> mtlb_axis('equal')
--> plot(J,U)
--> xtitle('arc cosinus')
--> xsetech([0.5 0.5 0.5 0.5])
--> mtlb_axis('equal')
--> plot(J,V)
--> xtitle('arc sinus')
```

On peut exporter ou imprimer les figures via le menu de la fenêtre de la figure.

V. Fonctions

1. Fonctions en ligne

La commande `deff([images]=nom_de_fonction(variables), 'expression de variables')` permet de construire une fonction en ligne.

Par exemple, si l'on souhaite pouvoir utiliser la fonction $g : \mathbb{R} \rightarrow \mathbb{R}$ sur les matrices, on déclare $x \mapsto \cos^2 x$

cette fonction en ligne par :

```
--> deff(' [y]=g(x)', 'y=(cos(x)).^2')
--> g(%pi)
ans =
    1.
--> g([0,%pi/4,%pi/2])
ans =
    1.    0.5    3.749D-33 (toujours pas 0!!!)
```

On peut évidemment créer de fonctions de plusieurs variables.

```
--> deff(' [u]=h(x,y,z)', 'u=sin(x+y.*z)')
--> a=%pi;
--> b=[-%pi:%pi/5:%pi];
--> c =[0:0.1:1];
--> h(a,b,c)
ans =
      column 1 to 6
    1.225D-16    0.2486899    0.3681246    0.3681246    0.2486899
1.225D-16
      column 7 to 11
   - 0.3681246   - 0.7705132   - 0.9980267   - 0.7705132   - 2.449D-16
```

2. Scripts

Plutôt que de taper les instructions de façon standard les unes après les autres avec les risques d'erreur que cela comporte, il est possible de les écrire dans un éditeur de texte et faire du copier-coller.

L'utilisation de l'éditeur de texte Scipad qui est intégré à Scilab (accessible directement depuis le menu de la console par `Fichier->ouvrir un fichier` ou `Applications->Editeur`) permet d'envoyer directement les commandes à la console. On peut aussi sauvegarder ces informations dans un fichier texte brut (sans mise en forme) avec l'extension `.sci`.

Pour pouvoir utiliser ce fichier, il faut que celui-ci soit dans le répertoire courant de Scilab. On peut obtenir cette information à partir du menu de la console : `Fichier->Afficher le répertoire courant`. On peut aussi modifier si nécessaire ce répertoire toujours par le menu de la console : `Fichier->Changer le répertoire courant`.

Pour exécuter ce fichier, on peut utiliser la commande `exec` dans la console en spécifiant le nom du fichier ou simplement via le menu de la dite-console (`Fichier->Exécuter`)

On peut ajouter des commentaires (par exemple "ne fonctionne pas") dans le fichier en préfixant ceux-ci par les sigles `//`.

Par exemple, on enregistre les 4 lignes suivantes dans un fichier `essai.sci`.

```
// un premier test
A=[1:6]
B=[4:9]
A*B'
```

L'exécution donne :

```
--> exec('essai.sci')
--> A=[1:6]
A =
    1.    2.    3.    4.    5.    6.
--> B=[4:9]
B =
    4.    5.    6.    7.    8.    9.
--> A*B'
ans =
    154.
Exécution terminée.
```

Les variables sont bien en mémoire après l'exécution du script.

3. Fonctions externes

Pour créer une fonction avec ou sans retour de valeur, on peut la placer dans un fichier toujours de texte brut avec l'extension `.sci`.

Il devra être enregistré dans un répertoire connu de **Scilab**.

Si l'on veut réutiliser la fonction $g(t) = (\cos(t))^2$, vue précédemment, dans des calculs ultérieurs, on crée un fichier nommé `fichier_de_g.sci` par exemple.

Copier votre fichier dans le répertoire courant de **Scilab** (on peut l'obtenir à partir de la console, Menu->Fichier->Afficher le répertoire courant) ou modifier celui-ci (Menu->Fichier->Changer le répertoire courant).

Dans ce fichier, on place les commandes :

```
function [y]=fonc_g(t)
y=(cos(t)).^2;
```

On peut ensuite appeler cette fonction dans **Scilab** :

```
--> getf('fichier_de_g.sci')
--> S=[0,%pi/4,%pi/2];
--> fonc_g(S)
ans =
    1.0000    0.5000    0.0000
```

On peut aussi définir des fonctions en ligne à l'intérieur de fonctions définies en externe.

```
function [z]=fonc_g(t)
def(' [y]=g(x)', 'y=(cos(x)).^2' )
z=g(t);
```

Dans le cas d'une fonction relativement simple, cela n'a guère d'intérêt.

On peut créer de la même façon des fonctions de plusieurs variables et même demander l'affichage d'une figure.

Par exemple, nous enregistrons dans un fichier `fonc_fpv.sci` la fonction suivante :

```
function [v] = fonc_fpv(x,y)
v=sin(x'*y);
xset('colormap',hotcolormap(128))
plot3d1(x,y,v)
```

De nouveau, il suffit ensuite d'appeler la fonction depuis Scilab.

```
--> getf('fonc_fpv.sci')
--> mtlb_close all
--> i=[0:0.05:2];
--> j=[0:0.05:3];
--> T=fonc_fpv(i,j);
```

-----Faire la feuille de TP n°2-----

VI. Programmation

Scilab intègre un langage de programmation. Ses commandes sont très classiques. Nous n'entrerons donc pas dans le détail de celles-ci.

Le point qui nous intéresse le plus est donc la syntaxe.

1. Opérateurs de comparaison, variables booléennes et opérations logiques

Les conditions des tests seront des variables booléennes qui pourront être obtenues par des opérateurs de comparaison ou le résultat d'opérations booléennes.

<code>==</code>	égalité
<code>>=</code>	supérieur ou égal
<code><=</code>	inférieur ou égal
<code><></code>	différent
<code>&</code>	"et" logique
<code> </code>	"ou" logique
<code>not</code>	négation

De plus, il existe des constantes booléennes : `%t` vrai et `%f` faux.

2. Opérateur conditionnel

La commande `if` est standard. Sa syntaxe est la suivante :

```
if <test1> then
    <instructions si le test est vrai>
elseif <test2> then
    <instructions si le test1 est faux et le test2 est vrai>
elseif <test3> then
    <instructions si le test1 est faux et le test3 est vrai>
...
else
    <instructions si le test1 est faux>
end
```

Les commandes `elseif` et `else` sont optionnelles.
Pour une écriture en ligne, il suffit d'ajouter des `,`.

3. Boucles

La mise en oeuvre de boucles peut se faire avec la commande `while` ou avec la commande `for` si l'on connaît à l'avance le nombre d'itérations.

```
while <condition>
    <instructions>
end
```

```
for i = imin : imax
    <instructions>
end
```

Une méthode élémentaire :

```
--> a=12
a =
    12.
--> b=18
b =
    18.
--> while abs(b-a)>0
--> if b<a then
--> a=b
--> else
--> b= b-a
--> end
--> end
b =
    6.
a =
    6.
```

Exemple de construction de matrice avec la boucle `for` :

```
--> clear
--> for i=1:3,for j=1:3, F(i,j)=i+(j-1)*3;end;end;
--> F
F =
    1.    4.    7.
    2.    5.    8.
    3.    6.    9.
```

La variable de la boucle peut prendre ses valeurs dans une matrice

```
--> A=[2 -2;3 4]
A =
    2.   -2.
    3.    4.
--> y=0
y =
    0.
--> for i=A, y=y+i, end
```

```
y =  
    2.  
    3.  
y =  
    0.  
    7.
```

3. Rapidité

Rappelons que **Scilab** est un logiciel dédié au calcul matriciel.

Lorsque le choix nous est offert, il est préférable de ne pas utiliser les boucles pour résoudre un problème.

Pour illustrer cette remarque, voyons deux commandes jumelles `tic` et `toc`. Elle correspondent à la mise en route et l'arrêt d'un chronomètre. Cela permet de vérifier le temps qu'a pris l'exécution d'une instruction ou d'une suite d'instructions.

Ecrire les séquences suivantes dans un éditeur de texte et faire un copier-coller successif de chacune d'elles dans **Scilab** :

Séquence 1

```
X=rand(100,100);  
Y=rand(100,100);  
tic  
Z=acos(3*X+2*Y);  
toc
```

Séquence 2

```
X=rand(100,100);  
Y=rand(100,100);  
tic  
for i=1:100  
    for j=1:100  
        Z(i,j)=acos(3*X(i,j)+2*Y(i,j));  
    end;  
end;  
toc
```

4. Animation des figures

Puisque les graphiques se superposent dans la fenêtre en cours, on peut créer des animations. La commande `pause(x)` permet d'effectuer, comme son nom l'indique, une pause de la durée `x` entre les exécutions des commandes d'affichage des figures.

```
--> y=[0:0.1:1.2];  
--> mtlb_axis([0,2,0,1.4]);  
--> plot2d(y,(y-1).^2,5)  
--> for x=[0:0.1:1.7]  
--> plot2d(x,(x-1).^2,-1)  
--> xpause(4000);  
--> end;
```

-----Faire la feuille de TP n°3-----