Feuille 1: Introduction à Scilab

(1) Dans Scilab on déclare les variables de manière assez naturelle, par exemple essayez:

```
alpha=1.54
b=1.32;
alpha
alpha;
b
```

(2) Pour représenter des matrices et des vecteurs, on utilise des commandes du type:

```
A=[1 2 3] ,  B=[2,3,4]
C=[1;3;5] ,  D=[1:4]
E=[1:10]  ,  F=[1:2:10]
G=[1:3:10],  H=[1:4:10]
I=[1:4]   ,  J=[1,2,3;4,5,6;7,8,9]
K=[3 4;alpha b]
G=K'      ,  H=C'
size(C)   ,  D(4)      , J(2,2)
```

Notons que si B est une matrice, B' est sa transposée. Si B est une matrice de taille $n*p$, $size(B)$ rend le vecteur $[np]$. Essayez aussi les commandes $size(B,'c')$ et $size(B,'r')$. De plus pour exhiber le coefficient $(1,3)$ de E, la commande est $E(1,3)$.

(3) Construire un vecteur x contenant les valeurs de 0 à 1 par pas de 0.1, Quelle est la dimension de ce vecteur?

(4) Les opérations addition/soustraction et multiplication sont obtenues de manière naturelle, pour l'opération a^b on utilise $a**b$ ou $a\hat{}b$. Essayez:

```
x=b+alpha
y=alpha-b
xx=alpha/x
yy=x*y
z=x**3
z=(x**2)**(1/2)
z=x**y
x^y
```

(5) Les constantes mathématiques telles que π ou i ou e sont obtenues avec $\%pi$, $\%i$ ou $\%e$. Essayez:

```
%i*%i
%e**(%i*%pi)
```

(6) Pour un nombre complexe c, le conjugué et le module (qui est la valeur absolue quand le nombre est réel) valent:

```
conj(c)
abs(c)
```

Essayez par exemple de calculer le module du nombre complexe $3 + 5i.$, celui de $2 + i$ et celui de leur produit.

(7) Les opérations élémentaires sur les matrices sont effectuées avec les symboles $+$,$-$ et $*$. Notons que les tailles des matrices doivent alors correspondre. Créez une matrice Z qui soit $2*3$ et une matrice Y qui soit de taille $3 * 2$. Calculer:

```
Z*A
2*Z
Z*Y
Y*Z
```

(8) Il existe d'autres opérations sur les matrices, comme $.*$ qui multiplie terme à terme les coefficients. Ainsi $A. * B$ donne la matrice dont les coefficients sont $A_{ij}B_{ij}$. Les opérations .+,.$-$ et ./ sont définies de la même manière. Comparez:

```
A=[1 2 3;1 5 6;7 8 9]
A.*A
A*A
A.^(3)
```

Si A est une matrice, $inv(A)$ ou $A\hat{}(-1)$ est son inverse.

(9) Deux matrices importantes sont *eye* et *zeros*, alors que la première contient des 1 sur la diagonale, la deuxième est remplie de zéros.

```
eye(1,7)
eye(8,1)
eye(2,3)
eye(3,2)
zeros(5,3)
zeros(3,2)
A
zeros(A)
```

(10) Nous allons maintenant écrire des fonctions en les sauvant dans des fichiers. Pour cela cliquez sur le bouton "Editor" et un éditeur de texte se lance. Créez un répertoire (par exemple `L3-EDO-TP }) et sauvegardez-y un fichier qui s'appelle par exemple $exemple-1.sci$.`

(11) Ouvrir le fichier *exemple* − 1.*sci* avec l'éditeur de Scilab ou un autre (*emacs* sous Unix, Bloc-notes sous windows...), et ecrivez le texte:

```
//premiere fonction
function r=poly(x)
  r=x^3-2*x+1
endfunction

//deuxieme fonction
function resultat=g(toto)
  resultat=2*toto
endfunction
```

Dans Scilab, lancer

```
getf ./td1/exemple-1.sci
poly(3)
g(1)
poly(%pi)
```

Entre les commandes $function$ et $endfunction$, on met d'abord la commande $r = poly(x)$. Dans ce cas r sera le résultat de la fonction et doit être défini dans le corps de la fonction. *poly* est le nom de la fonction et x le nom de la variable que l'utilisateur entrera quand il appelera la fonction. Dans la commande $poly(3)$, le programme lancera la fonction *poly* en remplacant partout x par 3. Les commentaires sont les lignes précédées de //.
La commande "getf" quant à elle permet de cherger le fichier, ici le fichier se trouve dans le sous-répertoire "td1" et s'appelle "exemple-1.sci". **Attention** , à chaque fois que vous modidifierez le fichier, il faudra le recharger avec la commande "getf".

(12) **Les boucles** for. A la suite de votre fichier exemple-1.sci, écrivez La fonction suivante qui calcule la racine carrée d'un nombre (grace à l'algorithme sumero-babylonien):

```
//algorithme sumero-babylonien de calcul de la racine carre
function x=sumbab(a)
  x=1
  n=5
  for i=1:n
    x=(x+a/x)/2
  end
endfunction
```

Testez cette fonction avec plusieurs exemples.

Règle importante: Remarquez que pour l'alignement des différentes instructions, on a suivi la règle suivante, très utile pour se repèrer dans le code: toutes les instructions qui sont à l'intérieur d'une boucle ou d'une fonction sont mises en retrait. C'est seulement avec le *end* correspondant que l'on reprend l'alignement précédent.

(13) **Les boucle** *while*. Toujours à la suite de votre fichier, tapez la fonction suivante:

```
//algorithme sumero babylonien de calcul de la racine carre
function x=sumbab2(a)
  x=1
  i=0
  while abs(x^2-a)>1.E-14
    x=(x+a/x)/2;
    i=i+1 //nombre d'iterations
  else //on imprime le resultat
    x
  end
endfunction
```

Testez cette fonction avec plusieurs exemples.

(14) **Structure if/then/else/elseif..** Dans un fichier "exemple-2.sci" tapez le code suivant :

```
function resultat=racine(a,b,c)
  d=b^2-4*a*c;
  if a==0
    resultat='degenere'
  elseif d>0 then
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    resultat=[x1 x2];
  elseif d==0 then
    x1=-b/(2*a);
    resultat=[x1 x1];
  else
    x1=(-b-%i*sqrt(-d))/(2*a);
    x2=(-b+%i*sqrt(-d))/(2*a);
    resultat=[x1 x2];
  end
endfunction
```

Que fait cet algorithme?