TP1 – Introduction à Scilab

Scilab est un logiciel de calcul scientifique développé par des chercheurs de l'INRIA. Il est disponible sous Linux, Mac, Windows, et est téléchargeable gratuitement à l'adresse

`http://www.scilab.org`

Scilab possède son propre langage de programmation, très proche de celui de Matlab. Il permet notamment de résoudre numériquement des systèmes linéaires (ou non-linéaires), des équations différentielles, des équations aux dérivées partielles, qui peuvent provenir de problèmes physiques. Il possède également un outil de visualisation graphique pour tracer des courbes, des surfaces, et ainsi visualiser des solutions d'équations différentielles, par exemple.

L'objectif de ce TP est de se familiariser avec l'environnement de Scilab et de comprendre les éléments basiques de son langage de programmation.

1 Calculer avec Scilab

Scilab peut être vu dans un premier temps comme une calculatrice scientifique. Lancer Scilab à partir du bureau. Le logiciel Scilab comporte une sous-fenêtre principale, la fenêtre de commande (ou 'Console'), dans laquelle on tape les instructions.

1. Dans la fenêtre de commande, taper successivement à la suite de la flèche `-->` les instructions suivantes, et comprendre ce qu'elles effectuent (on appuie sur la touche Entrée pour valider chaque ligne) :

```
6-10   4*(-5)   1/3    sqrt(9)   sqrt(8)   (%i)^2   (2+%i)^2
```

On peut également affecter des valeurs à des variables. Taper les instructions suivantes, et comprendre ce qu'elles effectuent :

```
x=2   x-3   sqrt(x)   x=x+4   y=x^2   x-y
```

2. Scilab permet également de manipuler des vecteurs. Dans la fenêtre de commande, taper les instructions suivantes et déterminer ce qu'elles effectuent.

```
x=[0,8,-1]    y=[%pi;1+%i;9]   y'          y.'          x+y'       x.*(y')

a=1:10        a=1:1:10         a=1:2:10    a=5:-0.5:-1  a.^2       cos(a)

a=5:-0.5:-1   size(a)          length(a)   a(1)         a(2)       a(3:5)

a(2:2:6)      a([1,3:7])       b=2*a       b(2:4)=0     a=a(3:$)   b=b(2:$-1)
```

Aide de Scilab. Scilab dispose d'une aide incluse dans le logiciel, que l'on peut consulter en pressant la touche F1. On peut également rechercher des informations sur une commande en tapant `help nom_commande` dans la fenêtre de commande, où 'nom_commande' est le nom de la commande souhaitée. Par exemple, taper `help size` dans la fenêtre de commande.

3. Manipulons à présent des matrices. Pour définir les matrices

$$A = \begin{bmatrix} -1 & 2 & 4 \\ 7 & 8 & -5 \\ 4 & 0 & 1 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 6 & -3 \\ 1 & -2 & 9 \\ 5 & 3 & -7 \end{bmatrix},$$

on peut taper les instructions suivantes :

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

3.1 Taper les instructions suivantes, et déterminer ce qu'elles effectuent :

```
C=[A,B]  D=[A;B]  size(C)  size(D)   length(C)   length(D)

E1=3*A   E2=3+A   E3=A*B   E4=A.*B   E5=A./B     E6=exp(A)

G=A*B    x=G(5)   y=G(2)   G(2,1:3)  G(2:3,1:3)  G(:)
```

3.2 Ecrire une commande qui permet de remplacer la deuxième colonne de la matrice A par son opposée.

3.3 Il existe également des commandes prédéfinies pour construire certaines matrices particulières. Taper les instructions suivantes, et déterminer ce qu'elle effectuent :

```
ones(3,1)   ones(1,4)   eye(5,5)   ones(3,3)   zeros(4,3)
```

3.4 Définir de la façon la plus simple possible les vecteurs et la matrice suivants :

$$\begin{aligned} v1 &= (1, 2, 3, 4, \ldots, 15, 16) \\ v2 &= (28, 26, 24, 22, \ldots, 14, 12) \\ v3 &= (1, 2, 4, 8, 16, \ldots, 128, 256) \\ A &= \begin{bmatrix} 1 & 2 & 4 & \cdots & 2^8 \\ 1 & 3 & 9 & \cdots & 3^8 \\ 1 & 5 & 25 & \cdots & 5^8 \end{bmatrix}. \end{aligned}$$

2 Boucles et fonctions

Il existe deux types de boucles dans Scilab : les boucles `for` et les boucles `while`. Nous allons les tester en écrivant les instructions dans un script que l'on éxécutera une fois les instructions tapées. Cette méthode est plus pratique que de taper toutes les instructions

dans la fenêtre de commande Scilab, surtout lorsque l'on veut éxécuter un grand nombre d'instructions.

4. Ouvrir l'éditeur de scripts Scilab 'SciNotes' via le menu Applications > SciNotes. Sauvegarder le script dans un dossier 'TP1'. Dans ce script, taper la suite de commande suivante :

```
x=0;
for k=1:4
x=x+k
end
```

Exécuter le script dans Scilab (en allant dans le menu Exécuter > Fichier avec écho), et interpréter le résultat qui s'affiche. On voit qu'à l'étape n de la boucle, le paramètre `k` vaut le coefficient n du vecteur `1:4`. Faire de même pour le script suivant (que l'on testera pour différentes valeurs de `p`) :

```
p=3;
for k=1:p
x(k)=1/k;
end
x
```

5. Construire à l'aide de boucles `for` les vecteurs v_1, v_2, v_3 ainsi que la matrice A de la question **3.4**. Pour la matrice A, on pourra utiliser deux boucles `for` imbriquées.

6. Contrairement aux boucles `for`, les boucles `while` éxécutent une suite d'instructions tant qu'une certaine condition est vérifiée ("while" = "tant que"). Copier la suite de commandes suivantes dans un script, l'éxécuter, et interpréter ce qui s'affiche :

```
x=0;
while x<10
x=x+1
end
```

Les boucles `while` sont particulièrement utiles lorsque l'on ne sait pas par avance le nombre d'étapes que comportera la boucle. Par exemple, l'algorithme suivant permet de calculer $\sqrt{2}$ à une précision de (au moins) 10^{-3} près :

```
x=2; //valeur proche de sqrt(2)
eps=10^(-3); //précision que l'on désire
while abs(x^2-2)>eps //on ne s'arrête que lorsque |x^2-2|<eps
x=x/2+1/x;
end
format(12); //nombre de décimales qu'on affiche après la virgule
x //solution approchée
sqrt(2) //solution exacte
```

Notons ici l'usage des `//` qui permet de commenter le code que l'on écrit : tout ce qui est écrit après ces `//` ne sera pas pris en compte dans l'éxécution du script. Il est indispensable

de commenter son code pour expliquer le sens de chaque ligne écrite, afin qu'on puisse le relire ultérieurement !

7. Scilab permet également de définir des fonctions pour effectuer automatiquement une série de tâches à partir d'arguments donnés. On peut par exemple coder des fonctions "classiques" avec la syntaxe suivante, que l'on écrira dans un script :

```
function z=f(x,y)
z=x^2+y^3;
endfunction
```

Exécuter ce script, puis taper successivement `f(1,2)`, `f(2,0)`, `f(1,1)` dans la console. On voit que la fonction `f` implémente la fonction de deux variables $(x, y) \mapsto x^2 + y^3$. On peut imaginer des fonctions plus complexes, comme celle que l'on a vu à la question précédente qui prend en entrée un $x_0 > 0$ et un $\epsilon > 0$ et qui rend en sortie une valeur approchée de $\sqrt{2}$ à ϵ près, en utilisant une itération partant de x_0. Elle prend alors la forme

```
function x=RacineApprochee(x0,eps)
x=x0; //première valeur dans l'itération
while abs(x^2-2)>eps //on ne s'arrête que lorsque |x^2-2|<eps
x=x/2+1/x;
end
endfunction
```

Taper ces commandes dans un script, puis l'éxécuter. Le tester ensuite en tapant dans la ligne de commande `RacineApprochee(2,10^(-3))`, `RacineApprochee(2,10^(-6))`.

8. Ecrire dans un script une fonction qui prend en entrée un réel $x > 0$ et qui rend en sortie la partie entière de x, c'est-à-dire le plus grand entier plus petit que x.

9. On peut également intégrer des distinctions de cas dans des fonctions à l'aide des commandes `if`, `then`, et `else`. Par exemple, tester les fonctions suivantes :

```
function y=ValAbs(x)
if x<0 then y=-x;
else y=x;
end
endfunction
```

```
function Y=remplace(X)
n=length(X);
for j=1:n
if X(j)<0 then Y(j)=0;
else Y(j)=X(j);
end
end
endfunction
```

Ecrire une fonction qui remplace les coefficients positifs d'un vecteur par leur carré et les coefficients négatifs par leur cube.