

## Travaux Dirigés 1

### Exercice 1

Afin de se familiariser avec l'environnement du logiciel, on exécutera dans Scilab les commandes qui suivent. A tout moment en faisant `help command` on peut avoir une aide sur la commande (ou supposée telle...) `command`.

- 1) Dans Scilab une variable est déclarée de manière naturelle, par exemple

```
alpha1 = 1.34
```

déclare la variable réelle `alpha1` et lui assigne la valeur 1.34. Scilab ne distingue pas entre les nombres 1 et 1.0, car il s'agit d'un logiciel de calculs numériques (et non pas de calculs formels). Les constantes mathématiques, telles que  $\pi$ ,  $i = \sqrt{-1}$  et  $e := \exp(1)$ , sont prédéfinies et obtenues par les commandes `%pi`, `%i` et `%e`.

- 2) Les opérations élémentaires sur les nombres réels ou complexes (addition, soustraction, division et multiplication) sont obtenues de manière naturelle, et l'opération  $a^b$  se note `a**b` ou `a^b` :

```
alpha1 = 3; beta1 = 4; gamma1 = 1 + 2*i;  
x = beta1 - alpha1  
y = alpha1 + beta1  
xx = alpha1/beta1  
yy = alpha1*beta1  
z = gamma1**3  
z - gamma1^3  
zz = beta1**(1/2)
```

Le conjugué d'un nombre complexe `gamma1` et son module (qui correspond à sa valeur absolue si `gamma1` est réel) sont obtenus par

```
conj(gamma1)  
abs(gamma1)
```

**Remarque importante :** certains noms, correspondant à des nom de fonctions mathématiques classiques, sont réservées et il ne faut pas les utiliser comme nom de variable. Ainsi, il ne faut pas nommer une variable `sin` (correspondant à la fonction sinus), `gamma`, la fonction Gamma d'Euler  $\Gamma(x) := \int_0^\infty t^{x-1} e^{-t} dt$ , ou encore `beta` (correspondant à la fonction Beta d'Euler  $B(x, y) := \int_0^1 t^{x-1} (1-t)^{y-1} dt$ ). Cependant les noms de variables tels que `beta1`, `gamma2` sont acceptés.

- 3) Un tableau de nombres (c'est-à-dire un vecteur ou une matrice) est représenté par des commandes du type :

```
A = [1, 2, 3]
```

```

B = [1 : 5]
C = B'
D = [6 ; 7 ; 8]
M = [11, 12, 13 ; 21, 22, 23 ; 31, 32, 33]

```

Noter que si B est un vecteur, B' représente son transposé. L'élément d'indice 2 du vecteur B est obtenu avec B(2), de même M(2,3) est l'élément sur la ligne 2 et la colonne 3.

- 4) Si A est une matrice de type [m, n] déclarée dans Scilab, alors A(1,:) représente la ligne 1 de la matrice et A(:,2) représente la colonne 2 de la matrice A (naturellement en supposant que  $n \geq 2$ ...). On peut ainsi accéder de manière simple à n'importe quelle ligne ou colonne de la matrice A. Comme entraînement, construire une matrice  $5 \times 5$  puis en extraire la 3-ème ligne et la 4-ème colonne.

- 5) Lorsque  $\text{debut} \leq \text{fin}$ , la commande

```
x = [debut:pas:fin]
```

crée un tableau à une ligne dont le premier élément est **debut**, le *pas* ou l'incrément est **pas** et l'élément final est **fin**, ou le plus grand multiple entier de **pas** inférieur à **fin**. Construire un vecteur x contenant les valeurs 0 jusqu'à 1 par pas de .01 ; quelle est la dimension de ce vecteur ?

Une autre commande utile est la commande `linspace` (pour faire penser de *linear space*-*ment*):

```
linspace(debut,fin,nombre_de_points)
```

Comparer par exemple les deux vecteurs `linspace(0,1,10)` et `linspace(0,1,11)`.

- 6) Créer un vecteur colonne de 11 points dans le plan complexe  $\mathbb{C}$  répartis sur le segment qui joint  $-i$ , et  $7 + i$ .
- 7) La matrice identité d'ordre n est désignée par `eye(n,n)`. Il faut noter qu'il y a aussi une matrice `eye(m,n)` pour tout entier  $m, n \geq 1$  qui correspond à la matrice dont les éléments diagonaux sont égaux à 1 et tous les autres égaux à zéro. Le vecteur `ones(m,n)` correspond à la matrice dont tous les éléments sont égaux à 1. Noter aussi que la commande `A = zeros(m,n)` déclare une matrice à m lignes et n colonnes dont tous les éléments sont nuls.
- 8) La commande `diag(A)` qui extrait la diagonale principale de la matrice A. Par ailleurs, par exemple, la commande `diag(-2:2)` crée une matrice diagonale  $5 \times 5$ . Essayer les commandes `diag(ones(4,1),1)` puis `diag(ones(4,1),-1)`, puis créer une matrice tridiagonale de taille  $5 \times 5$ .
- 9) Créer (rapidement...) dans Scilab la matrice A de taille  $10 \times 10$  telle que les éléments de la première et la dernière lignes, ainsi que ceux de la première et la dernière colonnes, sont égaux à 1, et tous les autres éléments sont égaux à zéro sauf ceux situés sur la diagonale principale qui valent 2.
- 10) Les deux opérateurs + et - s'appliquent aussi aux vecteurs et matrices de même taille. De manière générale, les opérateurs qui s'appliquent aux scalaires peuvent opérer aussi sur les

vecteurs en agissant sur chaque composante du vecteur. Par exemple si A et B sont deux vecteurs de même type leur somme et leur différence sont obtenues avec

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
B = [11, 2, 3; 4, 15, 6; 7, 8, 19];
A + B
B - A
```

- 11) Cependant il faut noter que pour le produit il y a deux notions distinctes :  $A*B$  donne le produit matriciel des deux matrices (lorsque cela est défini, c'est-à-dire lorsque le nombre de colonnes de A est égal au nombre de lignes de B), alors que  $A.*B$  donne la matrice dont les éléments sont les produits  $A_{ij}B_{ij}$  (on notera que ce produit n'est défini que si A et B sont de même taille).

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];
B = [11, 2, 0; 4, 0, 6; 0, 8, 19];
A*B
A.*B
A.^2
```

Lorsque A est une matrice carrée et n un entier positif,  $A^n$  représente la puissance n-ème de A. Si A est une matrice inversible  $\text{inv}(A)$  ou  $A^{-1}$  désigne son inverse. Noter aussi que si b est un réel,  $A.^b$  est le vecteur dont les éléments sont  $(A_{ij})^b$  (en supposant que ces derniers sont bien définis).

- 12) Pour obtenir la dimension d'un tableau (c'est-à-dire vecteur ou matrice) A on utilise l'opérateur `size` : ainsi `size(eye(2,3))` donne `[2,3]`.

---

### Exercice 2

La commande `rand(m,n)` donne une matrice de dimension  $[m,n]$ , à m lignes et n colonnes, dont les éléments sont des nombres (pseudo-)aléatoires, compris entre 0 et 1. Le déterminant d'une matrice carrée A est obtenu avec `det(A)`.

- 1) Soient  $A = 3*\text{eye}(3,3) + \text{rand}(3,3)$  et  $B = (A+A')/2$ . A quoi correspond la matrice  $D = B^{(1/2)}$  ? Calculer  $B-D^2$ .
- 2) Sachant que a et b sont deux vecteurs de dimension  $[1,n]$ , que signifie  $a*b'$  ?
- 3) En posant  $a = \text{rand}(1,20)$  et  $b = \text{rand}(1,20)$  calculer  $a*b'$ .

---

### Exercice 3

Dans Scilab on peut définir des fonctions utilisables par d'autres fonctions ou programmes. Pour cela on ouvre l'application SciNotes (dans l'un des menus de Scilab) et on crée un fichier, ou un script, avec l'un des suffixes `.sci` ou `.sce` dans lequel on écrit les commandes que l'on souhaite exécuter.

- 1) Dans SciNotes on peut définir une fonction par une commande du type

```
// définition de la fonction polyf
function resultat = polyf(x)
    resultat = x.^3 - 3*x.^2 + 1 ;
endfunction
```

où, entre les commandes `function` et `endfunction`, on place la déclaration `resultat = polyf(x)`, c'est à dire ce que la fonction `polyf` doit retourner, puis les différentes instructions permettant de définir la fonction `polyf`.

Un commentaire dans le programme, ignoré par Scilab, est noté par `// commentaire`). Noter aussi que SciNotes ajoute automatiquement `endfunction` lorsque l'on déclare `function`, et qu'elle met aussi en place une **indentation** permettant de visualiser les blocs d'instructions déterminant la fonction. De manière générale il est très important de respecter les indentations des diverses structures, lorsque l'on écrit un programme. N'oubliez pas que vous serez jugés également sur cet aspect...

Ces commandes seront contenues dans un fichier nommé `polyf.sci`, ou `polyf.sce`. Ensuite dans une fenêtre Scilab la commande

```
exec polyf.sci;
```

permet d'exécuter le code, de déclarer la fonction et de l'utiliser, par exemple en invoquant `polyf(1)`. On peut aussi exécuter le code à partir de la fenêtre SciNotes.

Pour vérifier votre degré d'attention, expliquer pourquoi il vaut mieux utiliser la déclaration

```
resultat = x.^3 - 3*x.^2 + 1 ;
```

plutôt que

```
resultat = x^3 - 3*x^2 + 1 ;
```

## 2) Lorsque les commandes suivantes

```
// polyfsquared.sci
// definition de la fonction
// f(z) = (z^3 - 3z^2 + 1)^2
function resultat = polyfsquared(x);
    exec polyf.sci; // fichier contenant la definition de polyf
    resultat = polyf(x)^2;
endfunction
```

sont mises dans le fichier `polyfsquared.sci`, et si le fichier `polyf.sci` se trouve dans le même répertoire, alors on peut utiliser la fonction `polyfsquared` après avoir chargé le fichier `polyfsquared.sci`.

## 3) Une suite d'instructions de Scilab peut se mettre dans un fichier ayant pour suffixe `.sce`, et alors on parle de *script* ou de programme de Scilab. Par exemple si le fichier `sumer.sce` contient les lignes suivantes

```
// sumer.sce
// algorithme sumero-babylonien de calcul de
```

```
// la racine carree de 2
a = 2 ;
x = 1 ;
n = 5 ;
for i = 1:n do
    x = (x + a/x)/2 ;
end
disp(x, "le résultat de la méthode sumérienne est x = ")
```

la commande `exec sumer.sce` exécutera le programme. Noter que lors de l'exécution d'un script les valeurs des variables ne sont pas nécessairement affichées sur la console. La commande `disp` correspond à l'anglais *display*, montrer.

- 4) Noter dans l'exemple précédent l'usage de la commande de boucle `for i = 1:n do...end`. L'indice utilisé dans la boucle parcourt un vecteur, et de ce fait on peut utiliser des incréments quelconques :

```
for i = 2.3:0.7:4
    i
end
```

ou même on peut faire circuler l'indice de boucle dans un vecteur quelconque prédéfini :

```
vect = [1, 4, 17, 6.5, 0];
for i = vect
    i
end
```

- 5) Des instructions conditionnelles de boucles peuvent être exécutées avec la commande `while`. Par exemple

```
// sumer_1.sce
// algorithme sumero-babylonien de calcul de
// la racine carree de 2
a = 2;
x = 1;
n = 0;
erreur = 1.E-14;
while abs(x^2 - 2) > erreur
    x = (x + a/x)/2;
    n = n + 1 // pour voir le nombre d'itérations
end
// on fait imprimer le resultat
// on affiche sur la deuxième ligne le nombre d'itérations
reponse = [x ; n]
disp(reponse, "Le résultat est ")
```

- 6) Il y a aussi la structure `if <condition> then`, par exemple

```

// racines.sci
// calcul des racines du trinome
// ax^2 + bx +c
function reponse = racines(a,b,c)
    delta = b^2 - 4*a*c;
    if a == 0 then
        reponse = 'polynome degenerate'
    elseif delta > 0 then
        x1 = (-b + sqrt(delta))/(2*a);
        x2 = (-b - sqrt(delta))/(2*a);
        reponse = [x1, x2];
    elseif delta == 0 then
        x1 = -b/(2*a);
        reponse = [x1, x1];
    elseif delta < 0 then
        x1 = (-b + sqrt(-delta)*%i)/(2*a);
        x2 = (-b - sqrt(-delta)*%i)/(2*a);
        reponse = [x1, x2];
    end
endfunction

```

---

#### Exercice 4

---

Pour représenter le graphe d'une fonction on utilise la commande `plot2d` ou `plot` : un exemple typique est

```

clf() ; // pour effacer la fenêtre du graphique
x = [0:0.1:2*pi]' ; // les points de discrétisation.
                        // Noter que x est un vecteur colonne
plot2d(x,sin(x))

```

Pour représenter plusieurs fonctions sur le même graphique, on peut procéder ainsi :

```

clf() ; // pour effacer la fenêtre du graphique
x = [0:0.1:2*pi]' ; // les points de discrétisation
plot2d(x,[sin(x), sin(3*x), sin(5*x)])

```

On peut faire des manipulations sur un graphique donné, mais nous ne nous y attarderons pas ici...